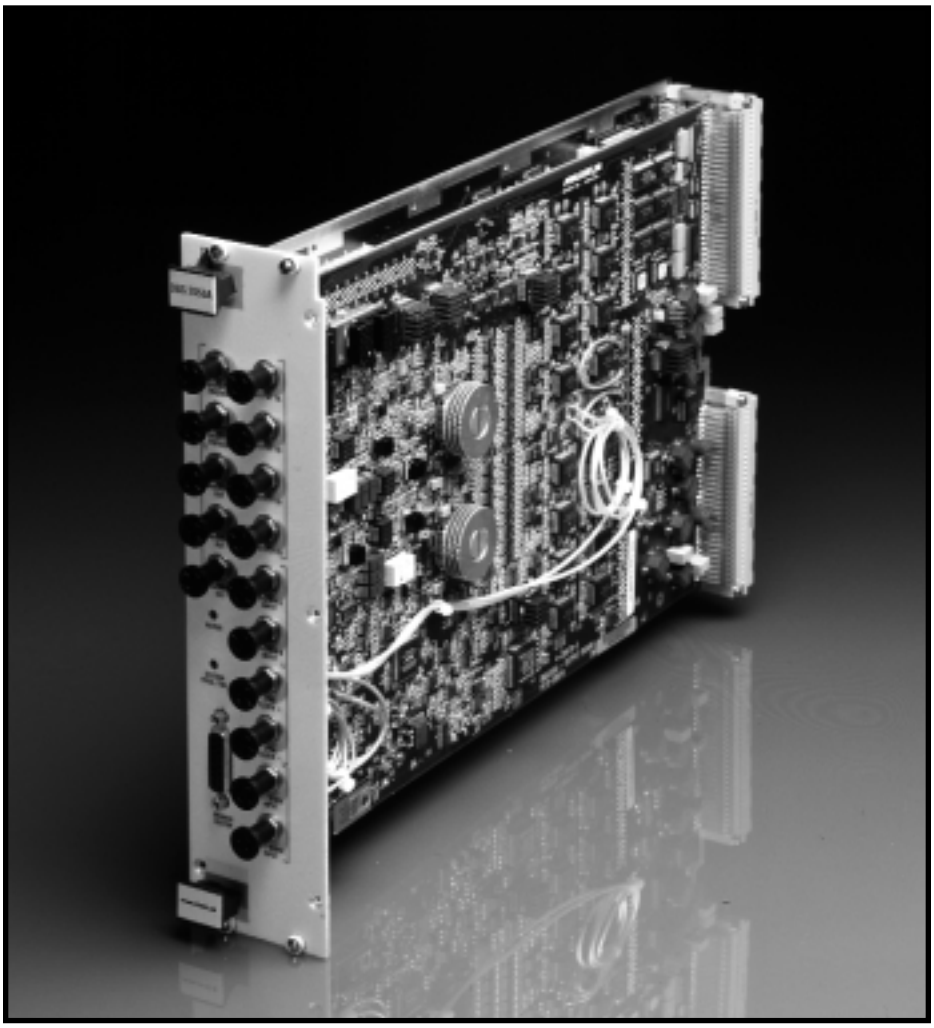
DBS2050A

ARBITRARY WAVEFORM GENERATOR



USER MANUAL

VXI bus

ANALOGIC®
The World Resource
for Precision Signal Technology

DBS2050A

2.4 GS/s Universal, High Speed Arbitrary Waveform Generator

User Manual

Revision 03

Proprietary Statement

The information contained in this publication is derived in part from proprietary and patent data of the Analogic Corporation. This information has been prepared for the express purpose of assisting operating and maintenance personnel in the efficient use of the instrument described herein. Publication of this information does not convey any rights to reproduce it or to use it for any purpose other than in connection with the installation, operation, and maintenance of the equipment described herein.

P/N 82-5126

Revision 03 January 2002

Printed in U.S.A.

Warranty

Analogic warrants only to the original purchaser that this product, as purchased from Analogic or an Analogic distributor or dealer, will conform to the written specifications for a period of one year from the date of purchase. If the product fails to conform to these warranties, Analogic, as its sole and exclusive liability hereunder, will repair or replace the product and/or its components within a reasonable period of time if the product is returned to a Analogic service center within the warranty period. These warranties are made upon the express condition that:

a) The purchaser shall promptly notify Analogic in writing of any non-conformity with the above warranty including a detailed explanation of the alleged deficiencies.

b) The product is returned to an Analogic service center at the buyer's expense after making suitable arrangements for performance of service.

c) When the product is returned for repair, a copy of the original bill of sale or invoice is sent with the product.

d) Analogic will not be liable for any incidental or consequential damages.

e) In the opinion of Analogic upon inspection, the product has not been misused, altered, or damaged due to abnormal handling and/or operation.

f) Repairs to the product and/or its components have not been made by anyone other than Analogic or one of its authorized repair agents.

g) The product has not been modified, altered, or changed in any manner by anyone other than Analogic or one of its authorized repair agents.

THIS WARRANTY EXCLUDES ALL OTHER WARRANTIES, WHETHER EXPRESSED OR IMPLIED, ORAL OR WRITTEN, INCLUDING WITHOUT LIMITATION WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.

No term, condition, understanding or agreement purporting to modify the terms of this warranty shall have any legal effect unless made in writing and signed by an authorized officer of Analogic and the purchaser.

Safety Precautions

Warnings and Cautions

The terms **WARNING** and **CAUTION** have specific meanings in this manual:

WARNING

A WARNING advises against certain actions or situations that could result in personal injury or death.

CAUTION

A CAUTION advises against actions that could damage equipment, produce inaccurate data or invalidate a procedure.

Mesures de securite

Mises en garde

Les terms **AVERTISSEMENT** et **ATTENTION** ont des sens spécifiques dans cette notice.

AVERTISSEMENT

Un AVERTISSEMENT informe des actions ou situations qui peuvent présenter un risque de blessure ou de décès.

ATTENTION

Une mise en garde débutant par le terme ATTENTION informe des actions que peuvent endommager le metériel, produire des données incorrectes ou nuire au fonctionnement.

Sicherheitsvorkehrungen

Warnung und Vorsicht

Die Bezeichnungen **WARNUNG** und **VORSICHT** haben in diesen Sicherheitsvorschriften eine besondere Bedeutung.

WARNUNG

Eine WARNUNG rät gegen bestimmte Handlungen oder Situationen, die Verletzung der Person oder Tod zur Folge haben können.

VORSICHT

VORSICHT weist auf Handlungen hin, die das Gerät beschädigen könnten, unrichtige Daten verursachen oder einen Vorgang auslöschen können.

SAFETY SUMMARY

The following safety precautions apply to both operating and maintenance personnel and must be observed during all phases of installation, operation, and service of the unit. Before applying power, follow the installation instructions and become familiar with the operating instructions for all components.

CAUTION

This product uses components which can be damaged by electrostatic discharge (ESD). To avoid damage, be sure to follow proper procedures for handling, storing, and transporting ESD-sensitive devices.

PRÉCAUTIONS

Ce produit utilise des composants pouvant être endommagés par décharge électrostatique (DES). Pour éviter les dégâts, suivre les procédures appropriées lors de la manipulation, du stockage, et du transport des dispositifs sensibles aux décharges électrostatiques.

VORSICHT

Dieses Produkt ist mit Komponenten ausgerüstet, die durch elektrostatische Entladung beschädigt werden können. Zur Vermeidung von Schäden muß dafür gesorgt werden, daß die maßgeblichen Vorschriften für die Behandlung, Lagerung und den Transport für elektrostatisch empfindliche Geräte beachtet werden.

CAUTION

Signal and communications cables should not be routed in troughs, wireways, pipes, or conduits containing power cables, nor routed close to electric motors, welding machines, or other equipment capable of generating significant amounts of electromagnetic interference (EMI). Failure to comply may result in improper system operation.

PRÉCAUTIONS

Les câbles de communications et de signaux ne doivent pas être acheminés par le biais de goulottes, de chenaux, de tuyaux, ou de conduits contenant des câbles d'alimentation. Ils ne doivent pas non plus être positionnés à proximité de moteurs électriques, de machines à souder, ou d'autres équipements capables de générer des quantités significatives d'interférence électromagnétique (EMI). Ne pas se conformer à ces règles pourrait avoir pour effet de dérégler le fonctionnement du système.

VORSICHT

Signal- und Kommunikationskabel sollten weder zusammen mit Leistungskabeln in Wannen, Leitungsbahnen, Rohren oder Kanälen verlegt werden, noch in der Nähe von Elektromotoren, Schweißmaschinen oder anderen Geräten, die beträchtliche elektromagnetische Beeinflussung (EMB) erzeugen. Nichtbeachtung kann zu Fehlfunktion führen.

CAUTION

Do not install substitute parts or perform any unauthorized modifications to this unit. Return the unit to Analogic for service and repair to ensure that the safety features are maintained.

PRÉCAUTIONS - NE PAS SUBSTITUER LES PIÈCES OU MODIFIER L'UNITÉ

Ne pas installer de pièces de substitution ni effectuer des modifications non autorisées sur l'unité. Renvoyer l'unité à Analogic pour l'entretien et les réparations, de manière à assurer que les dispositifs de sécurité soient maintenus.

VORSICHT - NUR ORIGINALTEILE BENÜTZEN UND DIE EINHEIT NICHT ÄNDERN

Es sind nur Originalteile einzubauen, und ohne Erlaubnis darf keine Änderung an der Einheit vorgenommen werden. Zur Wartung und Reparatur ist die Einheit an Analogic zurückzusenden, damit die Beibehaltung der Sicherheitseigenschaften gewährleistet ist.

Sales Support

If this product fails to operate satisfactorily upon arrival, contact **one** of the following to arrange for an exchange:

- Your Analogic dealer
- Test and Measurement Division (T&M) Order Entry at:

  Analogic
  Peabody, MA 01960
  Tel: (978) 977-3000 – Ext. 3840 or 3844
  FAX: (978) 977-6818

Upon contacting Order Entry, you will be given a Return Material Authorization (RMA) number. The RMA number is your authorization number.

Note: Please write this number, along with the product's serial number, on your purchase order and shipping label.

Send all authorized returns to:

Analogic Corporation
8 Centennial Drive
Peabody, MA 01960
Attn: Receiving Dock A
RMA# _____
Serial Number _____

Technical Assistance

For technical assistance, contact Analogic's Test and Measurement Division (T&M) Applications Engineering:

Phone:	**(978) 977-3000 or (800) 446-8936**	**Ask for Extension x-3260 or x-3232**
Fax:	**(978) 977-6814**	
E-mail:	**T&M_info@analogic.com**	

Our T&M Applications Engineers work with you to solve your problem. If the unit needs to be returned to the factory, the engineers refer you to our Customer Service Office.

Contents

PROPRIETARY STATEMENT II
WARRANTY III
SAFETY PRECAUTIONS IV
SALES SUPPORT VII
Technical Assistance ..... *vii*

1 INTRODUCTION 1-1

SCOPE OF THE MANUAL 1-1
RELATED MANUALS 1-1
DBS2050A OVERVIEW 1-1

2 PREPARATION AND INSTALLATION 2-1

OVERVIEW 2-1
UNPACKING THE DBS2050A 2-1
HARDWARE DESCRIPTION 2-2
Front Panel *2-2*
Logical Address *2-3*
Changing the Logical Address *2-3*
A24/A32 Communications *2-4*
HARDWARE INSTALLATION 2-5
Cooling and Power Requirements *2-5*
Installing the Module into a Chassis *2-5*
Extracting the Module from a Chassis *2-6*
SOFTWARE INSTALLATION 2-6
Computer Requirements *2-6*
CD Contents *2-6*
Installing the DBS2050A Software *2-6*
Installing Wavesmith *2-7*
Pre-compensation Option *2-7*
DBS2050A INSTALLATION VERIFICATION PROCEDURE – VISA 2-7
DBS2050A INSTALLATION VERIFICATION PROCEDURE – DP7040-2 OR DP7020-2 2-8
CALIBRATION 2-9
Factory Calibration *2-9*
User Calibration *2-9*

3 FUNCTIONAL DESCRIPTION 3-1

CONTROLLING THE DBS2050A 3-1
DBS2050A/2055 Soft Front Panel *3-2*
USER CALIBRATION 3-2
When to Perform User Calibration *3-2*
OPERATING MODES 3-3
Single Channel Mode Output Configuration *3-3*
Programming Example – Possible Single Channel Mode Configurations *3-4*
Dual Channel Mode Output Configuration *3-5*
Setting Up Dual-Channel Mode Output Configuration *3-5*

ANALOG SIGNAL PATH 3-6
Gain and Offset Control *3-6*
Gain and Offset Direct Mode *3-6*
Run-Time Parameters *3-7*
Differential Offset *3-7*
FILTERS 3-7
TIME BASE: SAMPLE CLOCK 3-7
Internal Sample Clock *3-7*
External Clock *3-8*
Reference Clock *3-9*
CREATING AND RUNNING WAVEFORM SEQUENCES 3-10
Waveforms *3-12*
Standard Waveforms *3-12*
Arbitrary Waveform Requirements *3-13*
Waveform Data Types *3-13*
Run-Time Parameters *3-13*
Segments *3-14*
Sequences *3-15*
Sequence Controls *3-16*
TRIGGERS 3-17
Main Trigger *3-17*
Advance Trigger *3-19*
Branch Trigger *3-19*
MARKERS 3-21
Marker Example *3-22*
Marker Behavior *3-23*

4 DBS2050A/2055 SOFT FRONT PANEL 4-1

OVERVIEW 4-1
INITIALIZING THE DBS2050A 4-2
Communication Protocols *4-2*
Launching the Soft Front Panel Application *4-3*
Selecting an Instrument *4-3*
INSTRUMENTS PANEL 4-4
Channel Mode *4-4*
Start New Session *4-5*
User Calibration *4-5*
Physical Setup for User Calibration *4-5*
User Calibration Procedure *4-5*
Read Temperature *4-6*
Show Details *4-7*
SIGNAL PATH 4-8
Port (Head) *4-8*
Output config *4-8*
Filter *4-9*
Direct Gain and Offset Mode *4-9*
Direct Gain and Offset Values *4-9*
Configuring the Output Signals *4-10*
CONTROLS PANEL 4-11
Sample Clock *4-12*
Reference Clock *4-12*
Main Trigger *4-13*
Advance Trigger *4-14*

Markers *4-15*
WAVEFORMS PANEL 4-16
Note about Pre-compensation *4-16*
Creating and Loading a Standard Waveform *4-17*
Changing the name of a loaded waveform *4-17*
Standard tab field descriptions *4-17*
Loading an Arbitrary Waveform from a File *4-19*
From File tab field descriptions *4-19*
SEGMENTS PANEL 4-20
Creating a Segment *4-21*
Segment Name: Format and How to Change *4-21*
Segment Panel Field Descriptions *4-21*
SEQUENCE PANEL 4-23
Creating and Running a Sequence *4-23*

5 DBS2050A/2055 "VXIPLUG&PLAY" DRIVER SOFTWARE 5-1

INTRODUCTION 5-1
Multi-module Support *5-2*
Instrument Control Parameters Maintained by the Driver *5-2*
Standard Waveform Data *5-2*
Waveform Data Memory Management *5-2*
Waveform, Segment and Sequence *5-2*
Branching *5-3*
Single Channel Mode and Dual Channel Mode *5-3*
Guidelines for Creating Applications using the DBS2050A VXIplug&play Driver *5-4*
Output a Waveform with specified run parameter set *5-4*
Output a Sequence with branches set for specific conditions *5-4*
DRIVER FUNCTIONS 5-6
INITIALIZE FUNCTIONS 5-6
Initialize ..... `an2050_init` *5-6*
Set Instrument Mode ..... `an2050_SetInstrumentMode` *5-9*
Choose Communication Method ..... `an2050_ChooseCommMethod` *5-11*
Perform a Self Test ..... `an2050_self_test` *5-12*
HARDWARE QUERY 5-13
Find All 2050's ..... `an2050_find2050Instrs` *5-13*
Scan VXI Chassis for 2050's ..... `an2050_scanChassis` *5-14*
Find VXI Instruments ..... `an2050_findVXIInstrs` *5-15*
Query Revision Number ..... `an2050_revisionQuery` *5-16*
Hardware Revision Query ..... `an2050_HWRevQuery` *5-17*
Verify Manufacturer and Model ID ..... `an2050_verifyID` *5-18*
Output Manufacturer and Model Ids ..... `an2050_VXIInstrID` *5-19*
Check if Valid or Invalid Session ..... `an2050_invalidSession` *5-20*
SHUTDOWN/RESET 5-21
Reset the 2050 ..... `an2050_reset` *5-21*
Reset and Test ..... `an2050_CResetNTest` *5-22*
Close 2050 ..... `an2050_close` *5-23*
Close All 2050As ..... `an2050_closeAll` *5-24*
WAVEFORM CREATION 5-25
Standard Waveforms *5-26*
Create Standard Wave ..... `an2050_CCreateStdWave` *5-26*
Create Standard Wave (I) ..... `an2050_CCreateStdWaveI` *5-28*
Create Standard Wave (T) ..... `an2050_CCreateStdWaveT` *5-29*

Create & Load Std. Wave ..... `an2050_CCrNLoadStdWave` *5-30*
Create & Load Std. Wave (I) ..... `an2050_CCrNLoadStdWaveI` *5-32*
Create & Load Std. Wave (T) ..... `an2050_CCrNLoadStdWaveT` *5-34*
Arbitrary Waveforms *5-36*
Load Waveform ..... `an2050_SLoadWaveform` *5-36*
Load Waveform (N) ..... `an2050_SLoadWaveformN` *5-37*
Load Dual Channel Wave2 ..... `an2050_SLoadDualChnlWave2` *5-39*
Load Dual Channel Wave2 (N) ..... `an2050_SLoadDualChnlWave2N` *5-40*
Load Dual Channel Wave Dup ..... `an2050_SLoadDualChnlWaveDup` *5-42*
Load Dual Channel Wave ..... `an2050_SLoadDualChnlWave` *5-43*
Load Waveform for DBS2055 ..... `an2050_SLoadWaveform_2055` *5-44*
Load Waveform for DBS2055 ..... `an2050_SLoadWaveformN_2055` *5-45*
Support *5-47*
Calculate Total Points ..... `an2050_CCalcTPts` *5-47*
Calculate Total Points (I) ..... `an2050_CCalcTPtsI` *5-48*
CLOCK COMMANDS 5-49
Internal and External Clock Functions *5-49*
Set Clock Source ..... `an2050_GSelClock` *5-49*
Set Currently Selected Clock Frequency ..... `an2050_GSetClockFreq` *5-50*
Get Clock Frequency ..... `an2050_GGetClockFreq` *5-51*
Get Clock Divider ..... `an2050_GGetClockDiv` *5-52*
Set Internal Clock Frequency ..... `an2050_GSetIntClockFreq` *5-53*
Set External Clock Frequency ..... `an2050_GSetExtClockFreq` *5-54*
Set External Source Clock Frequency ..... `an2050_GSetExtSrcClockFreq` *5-55*
Reference Clock Functions *5-56*
Select Reference Clock ..... `an2050_GSelRefClock` *5-56*
Set External Reference Clock Frequency ..... `an2050_GSetExtRefFreq` *5-57*
Low Level Clock Functions *5-58*
Set Gate Delay ..... `an2050_CSetGateDelay` *5-58*
Set Clock Divider ..... `an2050_GSetClockDiv` *5-59*
Set External Source Clock Divider ..... `an2050_GSetExtClockDiv` *5-60*
Set Internal Source Clock Divider ..... `an2050_GSetIntClockDiv` *5-61*
SIGNAL PATH CONTROL 5-62
Set Differential Offset ..... `an2050_CSetDiffOffset` *5-62*
Set Wave General Tolerance ..... `an2050_CSetWaveGenTol` *5-63*
Set Amplitude & Offset Direct On/Off ..... `an2050_CSetAmpOffsetDirectOnOff` *5-64*
Set Amplitude Direct Value ..... `an2050_CSetAmpDirectVal` *5-65*
Set Offset Direct Value ..... `an2050_CSetOffsetDirectVal` *5-66*
Set Filter ..... `an2050_CSetFilter` *5-67*
Configure Single Channel Output ..... `an2050_CConfSnglChnl` *5-68*
Configure Dual Channel Output ..... `an2050_CConfDualChnl` *5-70*
SINGLE AND DUAL CHANNEL MODE OPERATION 5-71
Single/Dual Channel Mode Select *5-72*
Set Single/Dual Channel Output Mode ..... `an2050_CSetChnlMode` *5-72*
Segment and Sequence Control *5-73*
Set Sequence (Start Sequencer) with Markers ..... `an2050_SSetSeqMN` *5-73*
Set Sequence (Start Sequencer) ..... `an2050_SSetSeq` *5-75*
Set Sequence (Start Sequencer) with Markers ..... `an2050_SSetSeqM` *5-76*
Stop Sequencer ..... `an2050_SStopSeq` *5-78*
Copy Sequence ..... `an2050_SCopySeq` *5-79*
Copy Sequence for DBS2055/2050AdualSync ..... `an2050_SCopySeq_2055N2050sync` *5-80*
Single Channel *5-81*

Define Segment ..... an2050_SDefineSeg 5-81
Define Segment for DBS2055/2050A Dual Sync .... an2050_SDefineSeg_2055N2050sync 5-83
Define Sequence ..... an2050_SDefineSeq 5-85
Define Sequence for DBS2055/2050A Dual Sync ... an2050_SDefineSeq_2055N2050sync 5-86
Define Sequence from Partial Sequence ..... an2050_SDefineSeqP 5-87
Define Sequence from Partial Sequence DBS2055 an2050_SDefineSeqP_2055N2050sync 5-88
Dual Channel 5-89
Define Dual Channel Segment ..... an2050_SDefineDualChnlSeg 5-89
Define Dual Channel Sequence ..... an2050_SDefineDualChnlSeq 5-91
Define Dual Chnl Seg by Duplicating Wave ..... an2050_SDefineDualChnlSegDup 5-92
Define Dual Chnl Seg. from Seg ..... an2050_SDefineDualChnlSegFromSeg 5-94
Define Dual Channel Segment1 ..... an2050_SDefineDualChnlSeg1 5-95
DBS2055 and DBS2050A Dual Synchronous Segment and Sequence Definition 5-97
Set and Start Sequence ..... an2050_SSetSeqM_2055N2050sync 5-97
Set and Start Sequence ..... an2050_SSetSeqMN_2055N2050sync 5-100
Run-Time Parameters 5-103
Define Run-Time Parameters ..... an2050_SDefineRunParms 5-103
Define Run-Time Parameters for DBS2055 ..... an2050_SDefineRunParms_2055 5-104
UNLOAD AND UNDEFINE 5-105
Unload 5-105
Unload Waveform ..... an2050_SUnloadWaveform 5-105
Unload Waveform for DBS2055 ..... an2050_SUnloadWaveform_2055N2050sync 5-106
Unload Sequence ..... an2050_SUnloadSeq 5-107
Unload Sequence for DBS2055 ..... an2050_SUnLoadSeq_2055N2050sync 5-108
Unload All Sequences ..... an2050_SUnloadAllSeqs 5-109
Undefine 5-110
Undefine Run Parameters ..... an2050_SUndefRunParms 5-110
Undefine Run Parameters for DBS2055 ..... an2050_SUndefRunParms_2055N2050sync 5-111
Undefine Segment ..... an2050_SUndefSeg 5-112
Undefine Segment for DBS2055 ..... an2050_SUndefSeg_2055N2050sync 5-113
Undefine Sequence ..... an2050_SUndefSeq 5-114
Undefine Sequence for DBS2055 ..... an2050_SUndefSeq_2055N2050sync 5-115
Undefine All Definitions ..... an2050_SUndefAllDefs 5-116
TRIGGER 5-117
Main 5-117
Set Trigger Mode ..... an2050_CSetRunTrigMode 5-117
Set Trigger Level ..... an2050_CSetTrigLevel 5-119
Advance 5-120
Set Advance Trigger Mode ..... an2050_GSetAdvanceTrigMode 5-120
Trigger Advance Trigger ..... an2050_GTrigger 5-121
Branch 5-122
Set Branch Vector with Markers ..... an2050_SSetBranchVectorMN 5-122
Set Branch Vector ..... an2050_SSetBranchVector 5-124
Set Branch Vector with Markers ..... an2050_SSetBranchVectorM 5-125
Set Branch Vector Source ..... an2050_GBranchVectorSrc 5-127
Set Branch Trigger Mode ..... an2050_GSetBranchTrigMode 5-128
Trigger Branch Trigger ..... an2050_GTrigger 5-129
MARKER 5-130
Set Markers ..... an2050_GSetMarker 5-130
Set Marker Mode ..... an2050_GSetMarkerMode 5-131
Configure Markers ..... an2050_GConfigMarkers 5-132
INFORMATIONAL 5-133

Query Number of Segments Available ..... `an2050_SSegsAvail` *5-133*
Query Waveform Data Memory Available ..... `an2050_SWaveMemAvail` *5-134*
Get Sequence Information ..... `an2050_SGetSeqInfo` *5-135*
Get Segments in Sequence ..... `an2050_SGetSeqSegs` *5-136*
Get Loaded Sequences ..... `an2050_SGetLoadedSeqs` *5-137*
Get Loaded Waves ..... `an2050_SGetLoadedWaves` *5-138*
Get Number of Sequences Loaded ..... `an2050_SGetNumSeqsLoaded` *5-139*
Get Number of Waveforms Loaded ..... `an2050_SGetNumWavesLoaded` *5-140*
Get State ..... `an2050_SgetState` *5-141*
Translate Error Code to Message ..... `an2050_errorMessage` *5-142*
Get Digital Board Revision & Serial Number ..... `an2050_CGetDigiBdRevNSN` *5-143*
Get Output Board Revision & Serial Number ..... `an2050_CGetOutputBdRevNSN` *5-144*
Get DBS2050A Serial Number & Calibration Date `an2050_CGetSNNCalibDate` *5-145*
Get Calibration Procedure Rev. and Document # .. `an2050_CGetCalibRevNDN` *5-146*
Get Digital, Output, Calibration Revision #s ..... `an2050_CGetRevisionNumbers` *5-147*
Get Digital and Output Board Serial #s ..... `an2050_CGetSerialNumbers` *5-148*
Get Calibration Procedure Document Number *5-148*

APPENDIX A DBS2050A PERFORMANCE SPECIFICATIONS 1

CLOCK AND TIMING 1
OUTPUT SIGNAL CHARACTERISTICS 2
OUTPUT SIGNAL AC SPECIFICATIONS 3
TRIGGER 4
WAVEFORM (SEGMENT, SEQUENCING, BRANCHING, AMPLITUDE, MARKERS) 5
CALIBRATION 6
SOFTWARE 6
VXI 6
ENVIRONMENTAL 6
NON-OPERATIONAL SHOCK & VIBRATION 7
MTBF PARTS RELIABILITY 7
EMC IMMUNITY 7
CONDUCTED EMISSIONS 7
RADIATED EMISSIONS 8
SAFETY 8
POWER 8
PHYSICAL 8

APPENDIX B PRE-COMPENSATION OPTION FOR DBS2050A 1

OVERVIEW 1
CONTENTS 1
INSTALLATION 2
Software Components *2*
Pre-compensation Library *2*
Pre-compensation Correction Data from Floppy Disk *2*
Hardware Installation *3*
PRE-COMPENSATION THEORY 4
Pre-compensation Library *4*
Theory of Operation – The Process of Pre-compensation *4*
FREQUENCY RESPONSE CORRECTION INPUT – PRE-COMPENSATION CORRECTION DATA ARRAY 6
SINC CORRECTION 6
BAND LIMIT ARRAY 6
HALF DC 6
CONVERT TO TIME DOMAIN 6
RE-SAMPLING 6

HANNING WINDOW 7
NORMALIZE DC 7
TIME DOMAIN CONVOLUTION 7
PAD TO LOOP TRANSFORM 7
Pre-compensating a Waveform Using the Soft Front Panel *8*
PROGRAMMING NOTES: CALLING THE PRE-COMPENSATION .DLL 10
Necessary Files *10*
Pre-compensation Function Prototype *11*
Parameter Descriptions *11*
USING THE PRE-COMPENSATION LIBRARY 12
Basic Rule to Follow when Pre-compensating a Waveform *12*
Example: Single Segment Sequence *12*
Looping *12*
One Shot *12*
Multiple Segment Sequence *14*
Overall Pre-compensation *14*
Multiple Segment Sequence *14*
Multiple Segment Sequence – One Shot – with Looping of segments within the Sequence *16*
FACTORS AFFECTING ACCURACY 18
Reduction of the padding *18*
Clipping *18*
Signal Bandwidth *19*
Aliasing *19*
Sampling Frequency Limitations *19*

List of Figures and Tables

Figure 1: DBS2050A Block Diagram 1-3
Figure 2: DBS2050A Front Panel 2-2
Figure 3: S1 Logical Address Switch (default shown) 2-4
Figure 4: A24/A32 Jumper Location and Settings 2-4
Figure 5: Channel Mode Block Diagram 3-3
Figure 6: DBS2050A Clock Diagram 3-9
Figure 7: Sequence Creation 3-11
Figure 8: Marker Example 3-22
Figure 9: Communication Protocols 4-2
Figure 10: Instruments Panel 4-4
Figure 11: Instruments Panel – Show Details 4-7
Figure 12: Signal Path Panel 4-8
Figure 13: Controls Panel 4-11
Figure 14: Waveforms Panel 4-16
Figure 15: Segments Panel 4-20
Figure 16: Sequence Panel 4-23
Figure 17: Cable Connections for Single Channel Mode Pre-compensated Wave Output 3
Figure 18: Waveforms Panel 8
Figure 19: Precompensation panel 10
Figure 20: Multiple Segment Sequence 15

Table 1: Front Panel 15 Pin Control Connector 2-3
Table 2: Power Requirements for DBS2050A 2-5
Table 3: Sample Parameter Definitions 3-4
Table 4: Direct Gain and Offset Values 3-6
Table 5: Standard Waveform Parameters 3-12
Table 6: Waveform Data Types 3-13
Table 7: Segment Parameters/Controls 3-15
Table 8: Sequence Parameters/Controls 3-16
Table 9: Main Trigger Parameters: Trigger Mode 3-17
Table 10: Main Trigger Source, Polarity and Level 3-18
Table 11: Advance Trigger Parameters 3-19
Table 12: Branch Trigger Parameters 3-20
Table 13: Marker Signals 3-21
Table 14: Marker Behavior in Sequencer Modes 3-23

1 Introduction

Scope of the Manual

This User Manual provides installation and operation instructions for the DBS2050A Arbitrary Waveform Generator. The information is presented in the following sections:

Chapter 1	**Introduction,** provides a brief overview of the product.
Chapter 2	**Preparation and Installation**, provides hardware and software installation instructions.
Chapter 3	**Functional Description**, provides information on product capabilities.
Chapter 4	**DBS2050A/2055 Soft Front Panel (SFP),** provides instructions for Soft Front Panel controls.
Chapter 5	**DBS2050A/2055 "VXIplug&play" Driver Software,** provides information about the DBS2050A driver functions, driver and operating system compatibility, and a list of all driver functions.
Appendix A	**DBS2050A Performance Specifications**
Appendix B	**Pre-compensation Option for the DBS2050A,** describes the capabilities and usage of the DBS2050A Frequency Response Pre-compensation Option.

Related Manuals

Other manuals supporting operation of this instrument are:

- DBS2055 4.8GS/s Arbitrary Waveform Generator User Manual
- Wavesmith User Manual

DBS2050A Overview

The DBS2050A is a 2.4GHz Arbitrary Waveform Generator (AWG) implemented as a two-slot C-Size VXI module. The DBS2050A includes the following capabilities:

- Single (2.4GS) or dual channel (1.2GS) operation modes
- Powerful looping, advancing and branching
- 8M samples Waveform Memory
- Gain and Offset on a waveform Segment basis

The high speed is achieved by clocking two D/A converters on opposite phases of a clock that has a frequency of half the output sample rate. The output of the two D/A converters is alternately sampled by a custom analog combiner ASIC to generate a sampled output at twice the D/A clock rate. This architecture provides the flexibility of both single and dual channel operation in a single instrument. Dual channel operation allows two output signals to be output at 1.2 GS with very precise alignment between the channels, critical to I/Q signal generation and other applications.

This concept is taken one step further with the DBS2055 4.8GS Arb. The DBS2055 can be configured as a single channel at 4.8GS, two at 2.4GS or four at 1.2GS, all with very precise alignment. The DBS2050A and DBS2055 use common hardware, driver and Soft Front Panel software to ease migration.

The DBS2050A provides 8M samples of waveform memory. When using dual channel operation, 4M samples per channel are available. Sequencing, branching and looping of up to 4096 memory segments allow very efficient use of memory and can be used to generate complex waveforms. Each segment consists of waveform data with sequencing information to allow looping that segment once, a number of times, or continuously, or to repeat until an Advance trigger event occurs to cause the sequencer to 'Advance' to the next segment. Each segment has settings for gain and offset control. This powerful capability can be used to do offset or amplitude sweeps very quickly with a single segment of waveform data.

Sequences contain one or more segments. Sequences can be set to loop continuously or run in one shot mode.

Looping, Branching and Advancing can be pre-programmed and then triggered via front-panel inputs or via the VXIplug&play driver software.

Triggering via software or external signal includes triggered start, stop, start/stop and gated modes.

Branch triggering allows up to 16 waveform sequences to be set up. Then an external branch vector and branch trigger can be used to arbitrarily select the sequence. This provides very fast external control over waveform generation.

Analog signal outputs are provided as differential or single-ended for one-channel operation, and single-ended when in two-channel mode. Output amplitude is 1Vp-p max. in X1 output or 4Vp-p max. when using the X4 output. Offset can be set to ±3.5V with 2mV resolution.

The internal sample clock is programmable over a range of 600 S/s to 2.4 GS/s. It is derived from a phase lock loop (PLL) that can be phase locked to an external reference clock. A front panel clock input is also provided.

Three separate marker outputs are available with programmable widths and delays.

Calibration parameters are loaded from EEPROM during initialization.. Yearly re-certification is recommended. In addition, a User Calibration utility, accessible from the DBS2050A/2055 Soft Front Panel software, is provided to allow re-calibration of the unit when necessary.

All functions of the DBS2050A are supported by the VXIplug&play Driver Software. The driver minimizes system integration and software development time. The DBS2050A also includes the DBS2050A/2055 Soft Front Panel application, which implements the most commonly used driver functions and serves as an example for driver usage.

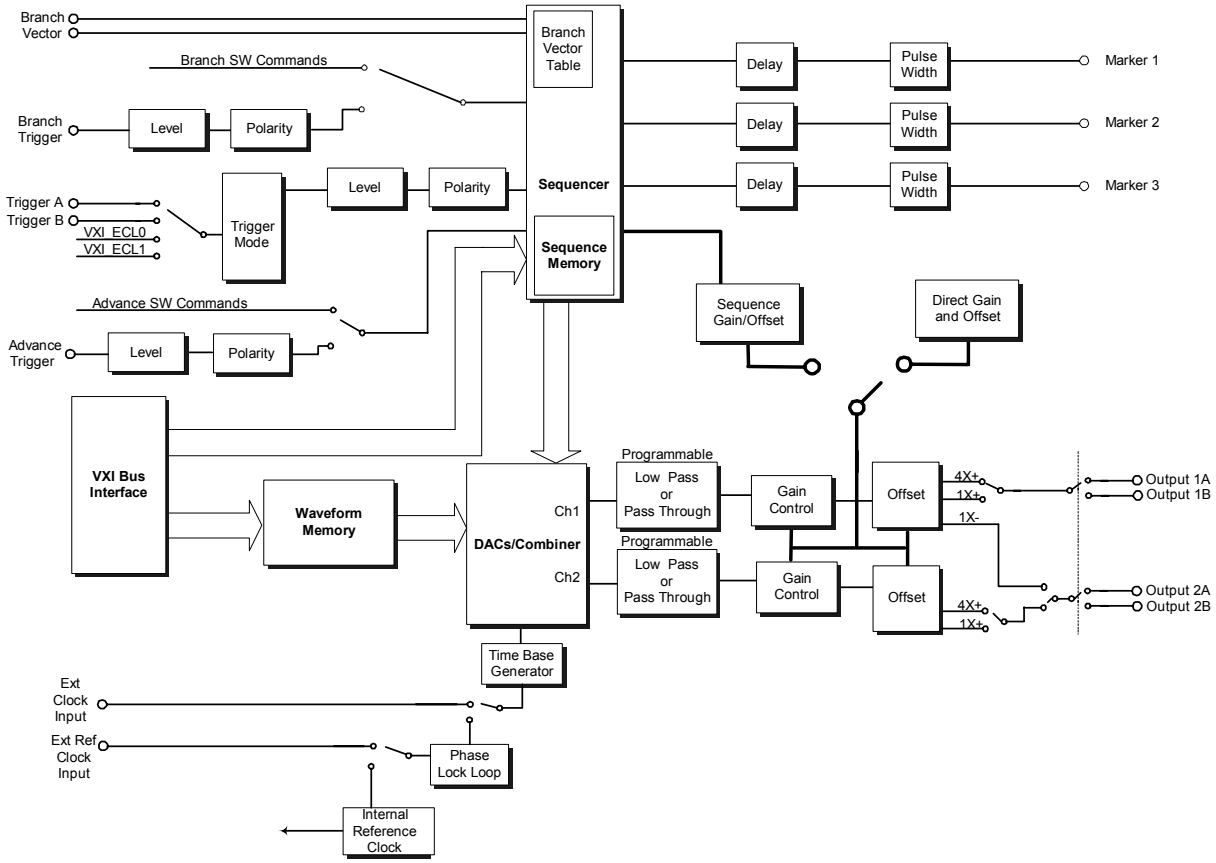


Figure 1: DBS2050A Block Diagram

2
Preparation and Installation

Overview

This chapter provides a packing list, a hardware description, instructions on installing the hardware into a chassis, and software installation instructions.

This chapter covers:

- Unpacking the DBS2050A
- Hardware Description
- Hardware Installation
- Software Installation
- DBS2050A Installation Verification Procedure – VISA
- DBS2050A Installation Verification Procedure – DP7040-2 or DP7020-2
- Calibration

Unpacking the DBS2050A

When you receive your DBS2050A, verify that the items listed below are included:

- DBS2050A module
- CD containing the DBS2050A/2055 Soft Front Panel (SFP) and software driver, as well as a copy of this manual in .pdf format.
- Pre-Compensation Kit, if purchased
- DBS2050A User Manual

Remove the items from the shipping container, carefully inspecting the hardware for any damage that may have occurred during shipment. If you notice any damage, contact Analogic. Save the shipping carton and packing materials in case you need to return the product for repair or replacement.

Once you have verified receipt of all of DBS2050A components and ensured the hardware was not damaged during shipment, you can begin setting up the DBS2050A hardware. This includes setting the logical address and communication mode, and installing the module into a VXI chassis.

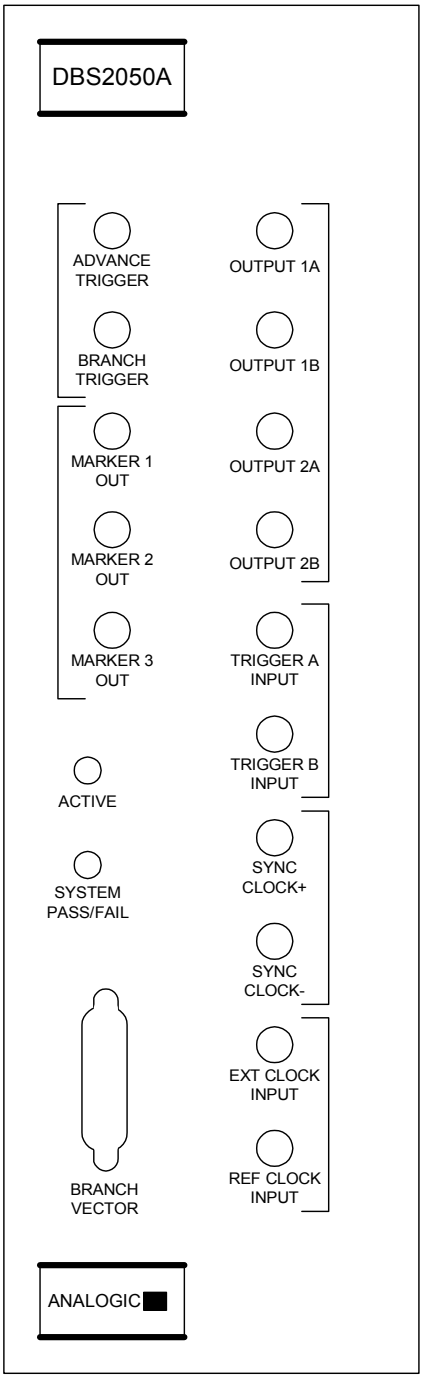
CAUTION

This product contains components, which are sensitive to electrostatic discharge (ESD). Be sure to follow proper procedures for handling, storing and transporting ESD-sensitive assemblies.

Hardware Description

The DBS2050A is a standard VXI Double Wide C-size module for use in any VXI-compliant chassis.

Front Panel



The Front Panel includes 15 SMA-type connectors, two status lights, and a 15 pin connector.

The SMA-type connectors are grouped as follows:

- **OUTPUT 1A&B, 2A&B**
  These are the main waveform outputs, see Figure 1: DBS2050A Block Diagram for a description of how they are used.
- **TRIGGER A&B INPUT**
  These are the main trigger inputs and either Trigger A or Trigger B can be selected.
- **SYNC CLOCK+/-**
  These are used to provide clock synchronization between modules on the DBS2055 and are not used on the DBS2050A.
- **EXT & REF CLOCK INPUT**
  Clock input is used to provide an external sample clock. Ref clock is used to provide an external reference clock.
- **ADVANCE AND BRANCH TRIGGER**
  These inputs are used to provide advance and branch trigger.
- **MARKER 1-3 OUT**
  These are TTL marker outputs used to provide timing signals synchronized to the waveform output.

The status lights represent the following:

- **ACTIVE**
  When this light is lit, it indicates that the Sequencer is in RUN mode. Either a waveform is running or is waiting to be triggered.
- **SYSTEM PASS/FAIL**
  When this light is lit, it indicates that the unit is powered up and has initialized successfully. If an error occurs upon startup or when an attempt to communicate with the host has failed, this light will be red.

Figure 2: DBS2050A Front Panel

The Branch Vector connector allows the input of a Branch Vector ID. This Branch Vector ID is used to identify which pre-defined Branch Sequence will be branched to upon a Branch Trigger event.

Table 1: Front Panel 15 Pin Control Connector

Pin	Description
1	Vector bit 0 LSB
2	Vector bit 1
3	Vector bit 2
4	Vector bit 3 MSB
5	NC
6	Reserved – do not connect
7	Reserved – do not connect
8	Reserved – do not connect
9-15	GND

Logical Address

Registers in VXI Bus devices are located in the A16 address space (within 64-byte blocks). The device's Logical Address is used by the software to communicate with these hardware registers.

The DBS2050A if shipped with Logical Address 128.

If you have only one DBS2050A module in the chassis, the Logical Address does not need to be changed. However, if you have more than one DBS2050A in a chassis or another VXI instrument at 128, each must have a unique Logical Address. Legal Logical Addresses range from 0 to 254 decimal.

Changing the Logical Address

The Logical Address is set via a set of 8 dip switches located on the bottom rear of the DBS2050A module and are accessed through a set of holes in the outer skin of the carrier.

To set the Logical Address:

1. Insert a needle nose probe (or similar tool) through the hole and push the dip switch to the desired setting.
2. Repeat for each additional dipswitch to be changed until the Logical Address is as desired.

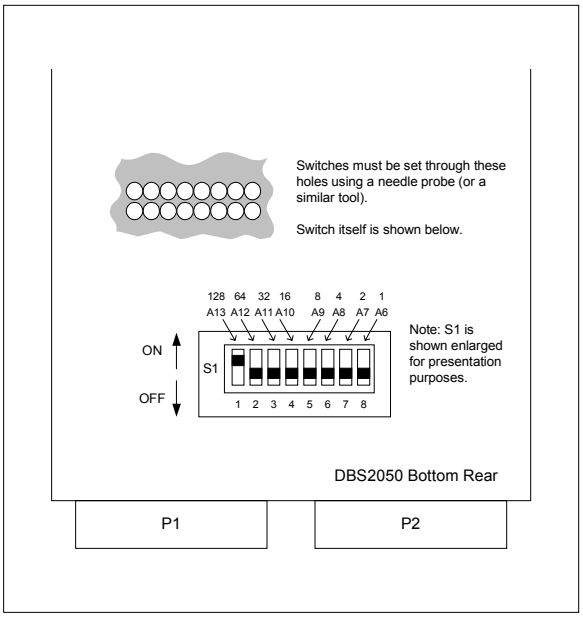


Figure 3: S1 Logical Address Switch (default shown)

A24/A32 Communications

The DBS2050A may operate in A24 or A32 communications mode.

A32 is the factory default, set via a jumper. If you have a slot 0 controller that supports A32 or are using Analogic's DP7040-2 or DP7020-2 chassis, keep the A32 jumper setting. If your controller does not support A32 mode, then use A24 mode.

To change to A24 mode, remove the jumper JP1.

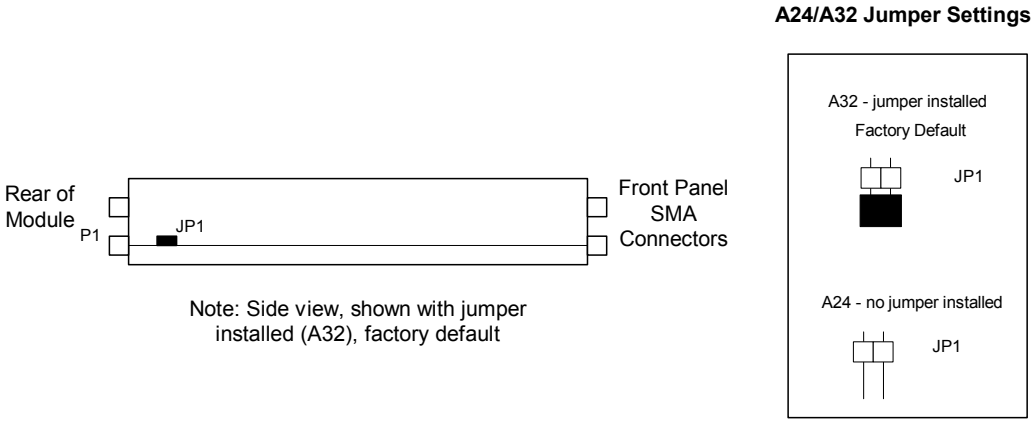


Figure 4: A24/A32 Jumper Location and Settings

Hardware Installation

Cooling and Power Requirements

Before installing the DBS2050A, verify that your VXI chassis meets the cooling and power requirements as specified in the tables below. Analogic's DP7040-2 and DP7020-2 chassis meets all cooling and power requirements for the DBS2050A.

The airflow requirement is 4.0 liters/second/slot minimum. The DBS2050A occupies 2 slots, for a total requirement of 8.0 liters/second minumum.

Table 2: Power Requirements for DBS2050A

Volts	Amps	Watts
24	0.1	2.4
-24	0.1	2.4
12	0.7	8.4
-12	1.0	12.0
5	6.2	31.0
-5.2	8.5	44.2
-2	3.6	7.2
Total		**108 Watts**

Installing the Module into a Chassis

A single DBS2050A module takes up 2 slots and can be installed into any VXI-compliant chassis or the Analogic DP7040-2 or DP7020-2.

To install the module:

1. Verify that the chassis power switch is turned OFF.
2. Insert the module into the desired slot by seating the module onto the slot's card guides and sliding it into the chassis until the pin connectors at the rear are connected. Firm pressure is required to seat the module properly.
3. Screw in the screws provided at the four corners of the module to secure the module to the chassis. If the screws do not take hold, remove the module and re-install.
4. Turn the chassis power switch ON. The DBS2050A System Pass/Fail LED should turn green.

Extracting the Module from a Chassis

The DBS2050A may be removed from the chassis by using the two extractors provided at the top and bottom of the module.

To extract the module:

1. Turn the chassis power switch OFF.
2. Unscrew the screws holding the module in place.
3. Place your thumbs on the top and bottom extractors and apply outward pressure until the module pops out.

Software Installation

The DBS2050A module is capable of interfacing with a host PC through a variety of VXI interfaces. The following sections describe the possible system configurations and the steps for installing the DBS2050A software.

Computer Requirements

Any host PC used with this software must meet the following minimum requirements:

- Pentium processor
- 32 MB RAM
- Windows 98/2000/NT 4.0 (Service Pack 3)
- 20 MB free hard disk space

CD Contents

The DBS2050A/2055 CD contains the following:

- DBS2050A/2055 Soft Front Panel
- DBS2050A User Manual in .pdf format
- DBS2050A/2055 VXIplug&play Driver
- Runtime engines for Labview and LabWindows, if necessary
- Platinum Communication Layer (PCL) for use with the Analogic DP7040-2 or DP7020-2 chassis

Installing the DBS2050A Software

To install the DBS2050A software:

1. Insert the DBS2050A/2055 software CD into your CD/ROM drive.
2. If the installation program does not automatically display:

   Go to My Computer and open whichever drive the CD is in.

   Double click on `setup.exe`
3. In the installation window, accept the default destination folder, C:\Vxipnp\Winxx\DBS2050A-2055, or click the Change button to enter a different path name. Click Finish.

After the installation process is complete, close the install script. A directory will appear showing all installed files.

The DBS2050A/2055 Soft Front Panel (SFP) may be accessed via the Windows Start Menu, along with online help.

Installing Wavesmith

Wavesmith is a software application allowing waveform creation, editing and analysis, and control of Analogic instruments.

Refer to the Wavesmith User Manual for more information about Wavesmith installation and operation.

Pre-compensation Option

If you ordered the Pre-compensation option, a diskette is included containing data necessary to pre-compensate a waveform. If you do not have this diskette, you will be unable to perform pre-compensation.

Pre-compensation data is unique for each specific DBS2050A and is generated at the factory prior to shipment. Please contact Analogic if you lose the floppy containing your pre-compensation data, or if you wish pre-compensation data to be generated for your module.

Refer to Appendix B, Pre-compensation Option, in this manual for more information.

DBS2050A Installation Verification Procedure – VISA

To verify the installation and setup of your DBS2050A in a standard VXI chassis containing a slot 0 controller using a Host PC running the Soft Front Panel:

1. Install the module into a chassis.
2. Connect the desired cables from the Output 1A SMA connector on the DBS2050A Front Panel to a scope of your choice.
3. Verify the VXI interface connection from the Host PC to the chassis controller (i.e, IEEE-1394 cable etc.).
4. Power ON the chassis.
5. Once the software has been installed, verify that the module is functioning properly by generating a standard sine wave using the DBS2050A/2055 Soft Front Panel.
    - Run *RESMAN* (accessible from the NI-VXI pull-down from the Windows Start Menu)
    - From the Start Menu, open the DBS2050A/2055 Soft Front Panel.
    - From the Initialization panel, select the desired communication layer – VISA or Platinum.

Select VISA for standard VXI chassis and Click *Find Instruments*.

- From the Available Modules table, select the desired module. Set Instrument Mode to DBS2050A and click OK to initialize the selected DBS2050A.
- On the Waveforms panel, click the Load Wave button to accept all default settings. The waveform name displays in the Loaded Waveforms table.
- Click the Quick Run button. When the Quick Run pop-up panel displays, click OK to accept the default settings.

  The DBS2050A should now be producing a 18.75MHz, 1Vp-p sinewave out of Output 1A.

DBS2050A Installation Verification Procedure – DP7040-2 or DP7020-2

This simple procedure will verify that a DBS2050A installed in an Analogic DP7040-2 or DP7020-2 is installed and functioning properly. The DBS2050A/2055 Soft Front Panel is used for this quick test, communicating with the chassis.

It assumes the DBS2050A is installed in a DP7040-2 or DP7020-2 chassis.

1. Connect from Output1A to a scope of your choice.
2. Verify the communications connection from the Host PC to the chassis (i.e., TCP/IP cable).
3. Power ON the chassis.
4. The DBS2050A/2055 Soft Front Panel (SFP) will be used to verify that the instrument is functioning properly.
   - From the Start Menu, open the DBS2050A/2055 Soft Front Panel.
   - From the Initialization panel, select the desired communication layer – Select *Platinum* and Click *Find Instruments*.
   - From the Available Modules table, highlight the DBS2050A instrument listed and click OK.
   - On the Waveforms panel, click the Load Wave button to accept all default settings. The waveform name displays in the Loaded Waveforms table.
   - Click the Quick Run button. When the Quick Run pop-up panel displays, click OK to accept the default settings.

     The DBS2050A should now be producing an 18.75MHz, 1Vp-p sine waveform out of Output1A.

Calibration

During initialization, the VXIplug&play driver applies calibration corrections automatically to the module. However, there may be times when it is necessary to re-calibrate the unit – for example, if there has been a drastic change in temperature. User Calibration is provided for this purpose.

Factory Calibration

Factory Calibration defines the register settings required for various timing delays and DC accuracy. These settings are stored in a Calibration Table in EEPROM. During operation, the DBS2050A uses these values from EEPROM to ensure accurate operation of the module.

The DBS2050A must be returned to the factory once a year for recalibration.

User Calibration

User Calibration may be invoked to improve DC accuracy. It is recommended that you run User Calibration when the temperature deviates more than 5 degrees Celsius from the temperature at which factory calibration was performed or the temperature of last user calibration. (The temperature of the module during factory calibration is printed in the Calibration Report.)

User Calibration may be invoked from the DBS2050A/2055 Soft Front Panel or from the driver.

Refer to the User Calibration section in Chapter 3 for detailed information about when and how to perform User Calibration.

3
Functional Description

This chapter provides a detailed description of the DBS2050A. It covers the full range of features and functions available to system integrators. It contains information essential to maximizing the unit's performance and flexibility. It assumes a basic understanding of AWG (arbitrary waveform generator) terminology and some programming experience.

This chapter covers:

- Controlling the DBS2050A
- User Calibration
- Operating Modes
- Analog Signal Path
- Filters
- Time Base: Sample Clock
- Creating and Running Waveform Sequences
- Triggers
- Markers

Controlling the DBS2050A

The DBS2050A can be controlled by:

- DBS2050A/2055 Soft Front Panel (SFP)
- Application program written by the user
- Wavesmith TM Waveform Generation and analysis software

All of these methods use the VXIplug&play Driver Software. The driver was developed using LabWindows CVI.

Driver functions are provided for all instrument control including:

- Clocks: Internal and External Sample and Reference Clocks
- Waveform generation and loading
- Sequence control
- Signal Path Control
    - Waveform Gain and Offset
    - Filters
    - Output Configuration (Single Channel X1, X4 or Differential X1, Dual Channel X1, X4)
    - Head A or B (also referred to as Port A or B)
- Triggers: Main, Advance and Branch
- Markers: Marker 1, Marker 2 and Marker 3

These controls are described in the following sections. For more information, refer to the section titled *Creating and Running Waveforms*, which describes waveform and sequencer operation.

DBS2050A/2055 Soft Front Panel

The DBS2050A/2055 Soft Front Panel application is an easy-to-use utility that allows the user to control the DBS2050A hardware. The SFP is also provided as an example for users creating their own applications. The SFP allows the user to quickly create and run standard and arbitrary waveforms and waveform Sequences.

The DBS2050A/2055 Soft Front Panel application controls all major functions provided in the driver except Branch Trigger.

More about the SFP can be found in the Soft Front Panel Chapter.

User Calibration

A User Calibration software utility is provided to perform user-initiated self-calibration of a DBS2050A. This utility is called from the SFP when the *Calibrate…* button is pressed on the Instruments panel. Hardware connections to the front panel do not affect DBS2050A user calibration.

It is useful to perform a user-calibration when the working temperature of the module at the user's site is different from the temperature during factory calibration or from the last User Calibration, due to different ambient temperature or airflow through the modules.

The DBS2050A/2055 User Calibration uses an internal ADC to measure the DC outputs of all the channels under all the different output configurations. The Gain and Offset errors are then stored in EEPROM. These values are subsequently used by the DBS2050A/2055 VXIplug&play Driver Software to correct the outputs.

When to Perform User Calibration

A User Calibration should be performed when the DBS2050A's current temperature exceeds the last calibration temperature, or the Factory calibration temperature (if User Calibration has never been performed), by more than 5ºC. Calibration should be perfromed only after reaching a stable temperature. The accuracy of the units is specified over a 5° Celsius change from the calibration temperature. Thus, in order to ensure that the module is within its performance specifications, User Calibration should be run when a 5°C temperature difference is exceeded.

Note: The Factory calibration temperature is printed on the DBS2050A Calibration report shipped with the module, under the title "END TEMP".

User Calibration may be run more often if higher accuracy is required.

To determine the temperature at the last user calibration, click the *Show Details* button on the Instruments panel. To determine the current temperature, click on the *Read Temperature* button from the Instruments panel.

Operating Modes

The DBS2050A is capable of operating in two output modes, Single Channel Mode and Dual Channel Mode.

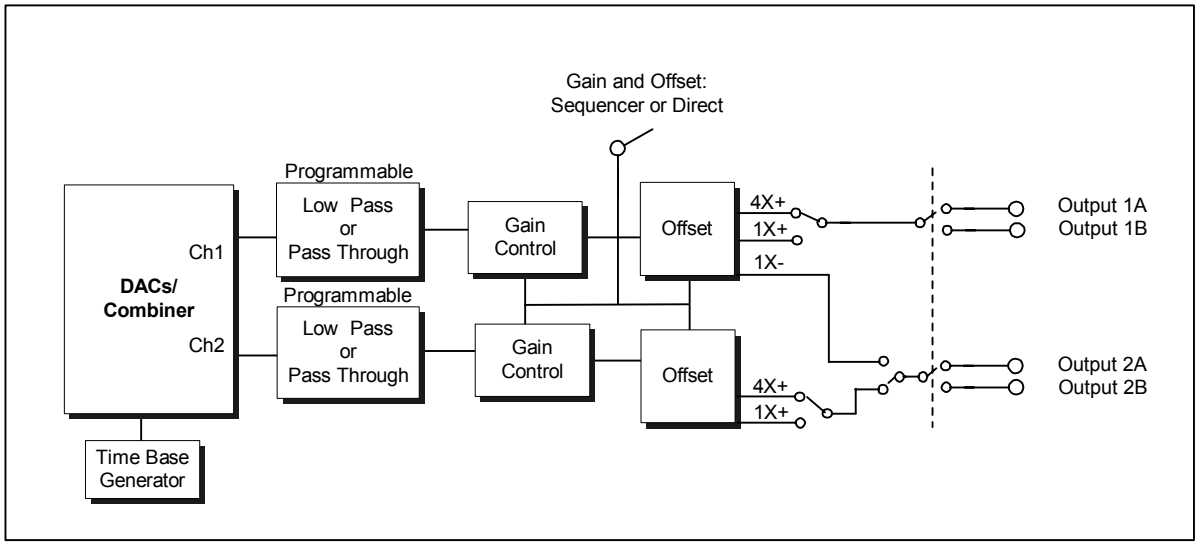


Figure 5: Channel Mode Block Diagram

Single Channel Mode Output Configuration

In Single Channel Mode, waveforms may be generated at sample rates up to 2.4GS/s. The waveform is produced through CH1 and is accessible from the SMA output connectors on the DBS2050A Front Panel.

For applications requiring the output of the DBS2050A to be applied to two different DUTs, two sets of switched output connectors are provided. These are referred to as Head A and B. Head A refers to the SMA connectors labeled Output 1A and Output 2A located on the Front Panel of the module. Head B refers to the SMA connectors labeled Output 1B and Output 2B located on the Front Panel of the module.

The output can be **Differential** or **Single-ended**:

- **Differential** CH1+ will be sent to Output 1 and CH1- will be sent to Output 2, at the specified Head (A or B).
- **Single-ended** Output (1X or 4X) can be sent to Output 1 at the specified Head (A or B). There is no signal to Output 2.

Programming Example – Possible Single Channel Mode Configurations

To set up Single Channel Mode:

1. Set the Output Mode to Single Channel using function `an2050_CSetChnlMode()`.
2. Configure Single Channel Output using function `an2050_CConfSnglChnl()` with parameters as shown in the table below.

Table 3: Sample Parameter Definitions

`an2050_CConfSnglChnl Parameter Settings`		
DESIRED OUTPUT	**PARAMETERS**	
	HEAD	CONFIG
Differential CH1+ to Output 1A CH1- to Output 2A	0 Head A	0
Differential CH1+ to Output 1B CH1- to Output 2B	1 Head B	1
Single-Ended X1 CH1+ to Output 1A No signal Output 2A	0 Head A	2
Single-Ended X1 CH1+ to Output 1B No signal Output 2B	1 Head B	2
Single-Ended X4 CH1+ to Output 1A No signal Output 2A (same as Single-Ended X1 except 4 times larger)	0 Head A	3
Single-Ended X4 CH1+ to Output 1B No signal Output 2B (same as Single-Ended X1 except 4 times larger)	1 Head B	3

Dual Channel Mode Output Configuration

In dual channel mode, even-indexed waveform data (the first data point = index 0) is output to Channel 1 and odd-indexed waveform data is output to Channel 2 at sample rates up to 1.2GS/s on each channel. This requires that waveforms be defined specifically for dual channel use. There are driver functions to aid in the creation of waveforms formatted properly for dual channel use (see the list below).

Setting Up Dual-Channel Mode Output Configuration

To set up Dual Channel Mode:

1. Set the Output Mode to Dual Channel using function `an2050_CSetChnlMode()`.
2. Configure Dual Channel Output using function `an2050_CConfDualChnl()`.

In Dual Channel Mode, the following parameters are available for output configuration.

Head A or B. Possible selections include the following scenarios:

- Output Head A for both Channel 1 and Channel 2 (Output 1A, Output 2A)
- Output Head B for both Channel 1 and Channel 2 (Output 1B, Output 2B)
- Output Head A for Channel 1, Output Head B for Channel 2 (Output 1A, Output 2B)
- Output Head B for Channel 1, Output Head A for Channel 2 (Output 1B, Output 2A)

All outputs can be X1 or X4.

Note: Differential output is not possible in Dual Channel Mode.

Analog Signal Path

The analog signal path processes waveform data from the DBS2050A. Signal path capabilities are waveform gain and offset control, selectable low pass filters, output (head) switching relays, single ended and differential output modes, and X1 or X4 output amplifiers.

Gain and Offset Control

The DBS2050A has 60db (1000:1) gain control range on each channel. X1 single-ended with a maximum output of 1.0Vp-p, X1 differential with a maximum output of 2.0Vp-p, and X4 single-ended with a maximum amplitude of 4.0Vp-p.

The offset range of each channel is ±3.5Vp-p. All voltages are specified into a 50 ohm load. In Differential output, this offset is applied as a common mode DC level to both outputs. There is also a differential offset mode described below.

There are two ways to control gain and offset of the waveform output: Directly under software control, or via Run-Time Parameters under control of the waveform sequencer.

Gain and Offset Direct Mode

Direct Gain and Offset allow immediate control of the waveform output. Direct Gain and Offset parameters are set as described in the table below.

Note that Direct control requires stopping and starting a sequence to change settings. This can be automatically handled by the driver.

Table 4: Direct Gain and Offset Values

Gain/Offset Parameter	Value
Direct Gain CH1 Direct Gain CH2	Specifies the desired gain in dB. The software will round off to the nearest achievable gain. Out-of-range values cause the gain to be set to the closest end. Valid range: 0 to -60dB.
Direct Offset CH1 Direct Offset CH2	Specifies the desired offset in Volts. Values outside the valid range are set to the closest end. Valid range: -3.5V to +3.5V.
Direct Gain and Offset ON/OFF	Specifies Direct Mode ON or OFF. If ON, Direct Gain and Offset values are used. If OFF, Run-Time Parameters are used for Gain and Offset.

Run-Time Parameters

The DBS2050A allows the gain and offset to be changed over the full range on a segment-by-segment basis while the waveform is being output. Each waveform segment has parameters for gain and offset of each channel. These are called Run-Time Parameters.

A user specifies one or more sets of Run-Time Parameters, and then associates one set with each Segment. When a Sequence is run, the Run-Time Parameters associated with each Segment in that Sequence are used to control the gain and offset.

Refer to the Creating and Running Waveform Sequences section for information on creating sequences.

Differential Offset

In Single Channel Mode with Differential output, Differential Offset may be applied. The Differential Offset value is defined below.

If the output mode is other than Differential, this value set is stored until the output mode is changed to Differential, at which time it takes effect.

DiffOffset (V)	Specifies the desired differential offset. Valid range: ±2.0V

Filters

Three low pass filters plus a no filter (flat response) setting are provided to reduce D/A aliases and to remove high frequencies. The filters are three pole Bessel filters. The settings are:

- No Filter or Flat Frequency response.
- 200 MHz cutoff.
- 20 MHz cutoff.
- 2 MHz cutoff.

Time Base: Sample Clock

The DBS2050A provides an internal sample clock source or can accept an external sample clock. The instrument operation for each is described as follows.

Internal Sample Clock

In internal clock mode a PLL based clock synthesizer is used to provide the low jitter sample clock. PLL frequency resolution (step size) is 5MHz at the highest frequency range of 1.2 GHz to 2.4GHz. Exact clock frequencies of 1.200, 1.205, 1.210…2.400GHz are available on the highest range. Lower frequency ranges are derived by dividing factors of two from the highest frequency range, i.e., 2400, 1200, 600, 300, 150MHz etc. are the maximum settings of each of the highest ranges. For

lower frequency ranges the step size also scales linearly. The highest nine frequency ranges and their step sizes are shown below.

Minimum Freq. (MHz)	Maximum Freq. (MHz)	Step Size (KHz)
1,200	2,400	5,000
600	1,200	2,500
300	600	1,250
150	300	625
75	150	312.5
37.5	75	156.25
18.75	37.5	78.125
9.375	18.75	39.0625
4.6875	9.375	19.53125

The DBS2050A driver calculates and returns the closest achievable frequency to the requested clock frequency.

When the internal sample clock is used, the user specifies the desired sample frequency in MHz. Sampling frequencies are from 600 to 2.4GHz for Single Channel Mode and 300Hz to 1.2GHz for Dual Channel Mode.

External Clock

An external clock can be used directly or can be divided internally to create the Sample Clock. The user can specify both the applied external clock frequency and the desired sample clock frequency. The driver then calculates the closest divider ratio to use.

Alternatively, the user can specify the external clock frequency and the desired divide ratio. Valid divider ratio are discrete integers and are not continuous. If the divider specified is not valid, the DBS2050A driver will snap to the nearest valid divider whose reciprocal is the nearest to 1/div. Valid dividers are 1, 2, 4, 8 to 64 in steps of 8, and from 64 to 4,194,394 in steps of 64.

The external clock frequency may be from 100kHz to 2.4GHz. Sampling frequencies may be specified from 600Hz to 2.4GHz for Single Channel Mode and from 300Hz to 1.2GHz for Dual Channel Mode.

When using an External Clock, the sample frequency value must be accurately provided to the driver. This allows the driver to set frequency-dependent calibration parameters.

Note: In Dual Channel Mode the sample rate is ½ the external clock rate. In Single Channel Mode the sample rate is equal to the external clock rate.

Reference Clock

When internal clock is used, the timing reference for the synthesizer can be either internal or external.

- Internal Reference Clock
  The internal reference clock has a frequency of 10 MHz.
- External Reference Clock
  If an external reference clock is used, its frequency must be specified. The applied frequency may be from 2.5MHz to 100MHz in 2.5MHz steps.

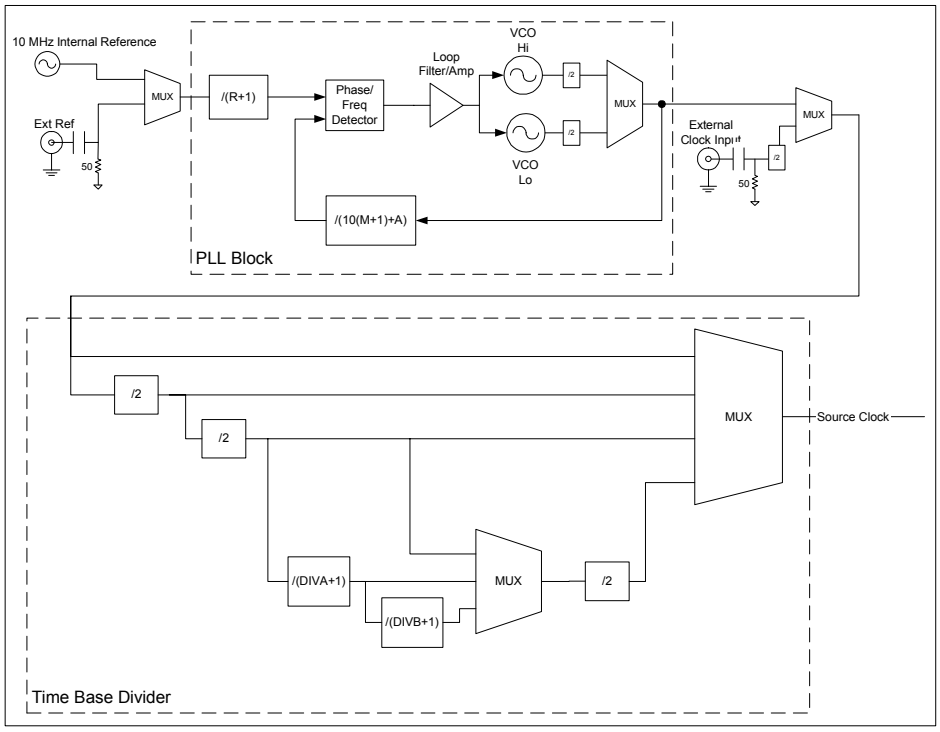


Figure 6: DBS2050A Clock Diagram

Creating and Running Waveform Sequences

The DBS2050A is a sequence-controlled waveform generator. This section describes the elements that make up a Sequence: Waveform data, Segments, and Run-Time Parameters.

A Sequence may be broken down into the following elements:

- Segments
  - Waveform data
  - Run-Time Parameters
  - Looping and Advancing instructions
- One Shot or Continuous instructions

A Sequence consists of one or more Segments and sequence controls that define whether the Sequence will run once (One Shot) or continuously (until a trigger event occurs or the Sequence is stopped).

A Segment consists of Waveform data, Run Time Parameters, and segment controls for looping, advancing, branching and markers.

Waveform data consists of raw 8 bit data which may loaded into Waveform memory using driver functions.

Run Time Parameters consist of gain and offset settings which are applied to each Segment.

The following figure describes how to build a waveform sequence.

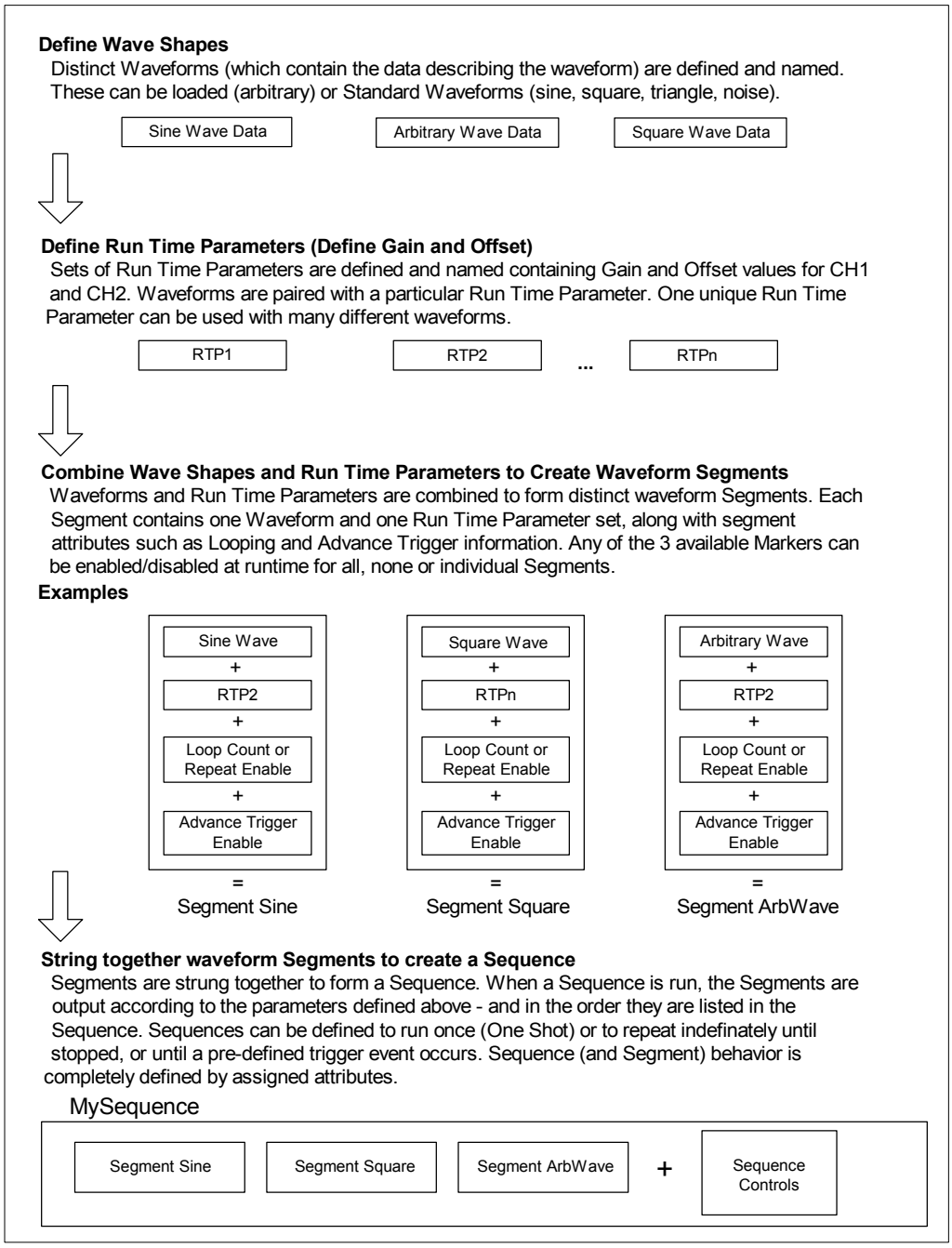


Figure 7: Sequence Creation

Waveforms

Waveform data is loaded into the DBS2050A Waveform Memory before it is used in a Segment. Waveform data can be entered:

- by defining standard waveforms (sine, square, triangle, noise waveforms), or
- by loading arbitrary waveform data from a file (DBS2050A/2055 Soft Front Panel) or via binary data array (VXIplug&play Driver functions).

When entered, Waveform data is stored internally and given a WaveID to be used when defining Segments. When using the Soft Front Panel, a descriptive name can be given to the waveform by the User. This name will appear in a list of available Waveforms.

Standard Waveforms

For standard waveforms, the following information must be provided.

Table 5: Standard Waveform Parameters

Waveform Parameters/ Controls	Description
Waveform Type	Specify the type of standard wave to be created. Possible types are: Sine Square Triangle Noise
Amplitude	Specify the amplitude of the waveform to generate. Must be between 0.0 and 1.0. Amplitude is relative to the full scale value of a signed 8 bit integer.
Points Per Cycle	Specify the number of points per cycle desired.*
Phase Angle	Specify the phase angle to begin with, in radians. This has no effect if the waveform type selected is Noise.

*See the section on Waveform Creation in the DBS2050A "VXIplug&play" Driver Software chapter.

Arbitrary Waveform Requirements

Waveform data must have a minimum of 512 data points.

The number of data points must be a multiple of 32 and at least 512 points.

In cases where the waveforms do not meet these requirements, the following options are available:

- The waveform data can be repeated until the requirements are met.
- The waveform data can be 'padded' with a value specified by the user – for example, the value of the last data point in the waveform, or some other desired value.

The DBS2050A/2055 Soft Front Panel allows the waveform to be repeated or to be padded using the last data point value.

Waveform Data Types

The VXIplug&play Driver accepts only an array of raw binary data. The SFP can accept various file formats. If Autoscale is enabled in the SFP, see the Soft Front Panel chapter for details on data formats.

Table 6: Waveform Data Types

	Driver Offset Binary	SFP (No Autoscale) 2's Compliment
+Full Scale	255	+127
0	128	0
–Full Scale	0	-127

Run-Time Parameters

Run Time Parameters allow waveform gain and offset changes on the fly. Each waveform segment has parameters for gain and offset settings of each channel.

In the VXIplug&play Driver, a user specifies Run-Time Parameter sets, then associates one set of parameters with each Segment. This way a set of gain settings can easily be used on multiple segments. When a Sequence is run, the Run-Time Parameters associated with each Segment in that Sequence are used to control the gain and offset.

When a Sequence is run in Single Channel Mode, Run-Time Parameters for Channel 2 are ignored.

The following information is contained in each set of Run-Time Parameters:

Channel 1 Gain	Specify the gain for Channel 1. Valid range: 0 to –60dB.
Channel 1 Offset	Specify the offset for Channel 1. Valid range: -3.5V to +3.5V.
Channel 2 Gain	Specify the gain for Channel 2. Valid range: 0 to –60dB.
Channel 2 Offset	Specify the offset for Channel 2. Valid range: -3.5V to +3.5V.

Run-Time Parameters are stored with a waveform segment. When running a Sequence, Run Time Parameter gain and offset values may be overridden by using Gain and Offset Direct Mode. When Gain and Offset Direct Mode is ON, direct Gain and Offset values for each channel are applied instead of the Run-Time Parameters.

Segments

A Segment associates a set of waveform data with a set of Run-Time Parameters. In addition, it is within a Segment that looping, advance trigger and markers are enabled/disabled. A Segment may be set to repeat continuously until a trigger event occurs or to run for a specific loop count.

When defined, Segments are stored and given a SegID to be used when defining Sequences. When using the Soft Front Panel, a descriptive name can be given to the Segment by the user. This name will appear in a list of available Segments when defining Sequences.

Caution:
To avoid a ‘trapping’ condition when Repeat Loop is enabled, be sure that Advance Trigger is not disabled (set to ignore Advance Trigger). Otherwise, the segment will run indefinitely until the entire Sequence is terminated.

The following information is specified in a Segment.

Table 7: Segment Parameters/Controls

Segment Parameters/ Controls	Description
Waveform	Specify a Waveform from the previously defined list of available Waveforms. In the VXIplug&play Driver, these are referenced by the assigned WaveID. On the DBS2050A/2055 Soft Front Panel, the Waveform Names are listed for selection.
Run-Time Parameter Set	Specify a set of Run-Time Parameters to be associated with the Segment. In the VXIplug&play driver, these are referenced by the assigned RTParmID. On the DBS2050A/2055 Soft Front Panel, the Run Time Parameter Names are listed for selection. When the Sequence is run, these parameters can be overridden by using the Direct Gain and Offset values, without affecting the segment definition.
Repeat Loop	Enabled – the segment repeats continuously. Loop Count is ignored. Disabled – the segment repeats the number of times specified in Loop Count.
Loop Count	The number of times the segment is repeated. This field is only valid if Repeat Loop Continuous is disabled.
Advance Trigger	Enabled – the segment repeats until an 'Advance to next segment' trigger occurs. This setting overrides Repeat Loop and Loop Count for this segment. Disabled – 'Advance trigger' is ignored.
Marker 1,2,3 Enable	See Marker section.

Sequences

A Sequence consists of one or more Segments linked together and run in one continuous operation. It also defines how Segments progress.

One or more Segments are added in the order in which they are to play. The waveform will run its Segments in the order they appear in the Sequence (except when Branch trigger is used).

When defined, Sequences are stored and given a SeqID to be used when running waveforms. When using the Soft Front Panel, a descriptive name can be given to the Sequence by the user. This name will appear in a list of available Sequences when selecting a waveform to be run.

The following information is specified in a Sequence:

Table 8: Sequence Parameters/Controls

Sequence Parameters/ Controls	Description
Segment List	List (array) of Segments to be included in the Sequence.
Loop Mode	The Sequence is to loop continuously (until a trigger event occurs or until the Sequence is terminated), or to be run once (Single Shot).

Sequence Controls

A Sequence can be defined to run continuously or to run once and stop. Single-Shot Mode refers to a Sequence that will run through once and stop. Each segment in the Sequence will be run according to its definition.

The single-shot mode can be used with the external start trigger or internal trigger.

In external start trigger mode, the first sample of the waveform starts after the desired trigger edge and proceeds until the last data sample. After the last sample, the output voltage goes to a midscale DAC value (zero) while holding constant the last gain and offset setting. After a holdoff time, the sequencer restarts at the beginning of the waveform sequence and waits for a new trigger edge. If the holdoff time is violated, and a trigger edge happens while the sequencer is restarting or while the waveform is in progress, the trigger will be ignored.

In internal trigger mode, the external trigger is ignored and the waveform starts as soon as the sequencer arms it. The waveform sequence runs to the last sample and then the output goes to a midscale DAC value (zero) while holding constant the last gain and offset setting. At this time the waveform does not re-arm or restart.

Triggers

This section describes the three types of triggers available in the DBS2050A:

- Main Trigger – used to initiate and optionally stop a sequence
- Advance Trigger – used to 'advance' from one segment to the next within a sequence
- Branch Trigger – used in conjunction with the branch vector to arbitrarily jump or branch to one of sixteen sequences

Main Trigger

The Main Trigger is used to initiate and optionally stop a sequence. It is configured by setting the parameters described in the following table.

Table 9: Main Trigger Parameters: Trigger Mode

Main Trigger Mode	Description
Free Run	In Free Run mode, the first sample of the waveform starts after the sequencer arms and proceeds indefinitely as the waveform data sequence is run.
Triggered Start	In Start trigger mode, the first sample of the waveform starts after the desired trigger edge and proceeds indefinitely as the waveform data sequence is run. The waveform is stopped under program control.
Triggered Stop	In Stop trigger mode, the first sample of the waveform starts after the sequencer arms the trigger circuit. The output waveform proceeds until the desired edge of the trigger occurs. At this time, the output voltage holds the last value of the waveform data and the last valid Gain and offset DAC values.
Gated Trigger	In Gated trigger mode, the first sample of the waveform starts after the desired (rising or falling) trigger edge and proceeds until the opposite edge of the trigger occurs. At this time, the output voltage holds the last value of the waveform data. The sequencer restarts at the beginning of the sequence, and re-arms the trigger circuit. When the original trigger edge occurs again, the waveform is restarted

	from the beginning of the sequence
Triggered Start/Stop	In Start/Stop trigger mode, the first sample of the waveform starts after the desired trigger edge and proceeds until the same edge of the trigger occurs. At this time, the output voltage holds the last value of the waveform data and the last valid Gain and offset DAC values. The sequencer restarts at the beginning of the sequence, and re-arms the trigger. When the same trigger edge occurs again, the waveform is restarted from the beginning of the sequence.

Table 10: Main Trigger Source, Polarity and Level

Main Trigger Parameter	Description
Main External Trigger Source	Specifies whether the trigger signal comes from the front panel SMA connectors (Trigger A or Trigger B), or from the VXI bus (ECL 0 or ECL 1).
Main Trigger Level	Specifies the trigger threshold voltage for the Front panel SMA Triggers. The value can be between ± 10.0 volts.
Main Trigger Polarity	Specifies whether to trigger on a negative or positive slope (rising or falling edge of the waveform).

Advance Trigger

Advance Trigger is used to *advance* to the next segment in a Sequence.

Advance trigger is enabled or disabled on a segment by segment basis. When Advance Trigger is enabled on a particular Segment and a Sequence containing that Segment is run, that Segment will loop continuously until an Advance Trigger occurs or the Sequence is terminated.

The following parameters configure the Advance Trigger.

Table 11: Advance Trigger Parameters

Advance Trigger Parameter	Description
Source	Specifies whether the trigger signal comes from the front panel SMA Advance connector or from a software command via the Advance Trigger software function.
Level	Specifies the trigger level as either 1.5 Volts (TTL) or zero volts.
Polarity	Specifies whether to trigger on the negative or positive slope (rising or falling edge of the trigger signal).

Branch Trigger

Branch Trigger is used in conjunction with the branch vector to arbitrarily jump or branch to one of sixteen sequences. The Branch vector is either a four bit TTL signal applied to the Branch Vector Connector, or can be an internally software generated value. The internal software branch is not arbitrary, but always jumps to the next branch vector in the table.

The Branch Trigger then is used to initiate branching to the selected sequence. When a branch trigger event occurs the current Sequence terminates and a branch is made to the beginning of the designated Sequence.

The Branch Trigger is configured via the following parameters.

Table 12: Branch Trigger Parameters

Trigger Parameter	Description
Branch Vector, SeqID and ithSeg	Valid range for Branch Vector: 0 to 15 Also, specifies the Sequence (and which Segment to branch to within the Sequence).
Branch Trigger Source	Specifies whether the trigger signal comes from the front panel SMA Branch Trigger connector or via the Branch Trigger software command.
Branch Vector Source	Specifies whether the Branch Vector comes from the Branch Vector input connector on the Front Panel of the DBS2050A (4 bit TTL) or from software commands. When software controlled, the branch vector will Auto-Increment (increment through 16 pre-defined branch vectors sequentially)
Branch Trigger Level	This setting is used if the source is the front panel SMA. Specifies the trigger level as either 1.5 Volts (TTL) or zero volts.
Branch Trigger Polarity	Specifies whether to trigger on a negative or positive slope (rising or falling edge of the waveform).

Markers

The DBS2050A has the capability to output 3 independent marker signals. Marker signals are intended to provide a means of triggering external events synchronously with the main output signal. The markers are called MARKER 1, MARKER 2 and MARKER 3. These signals may be accessed via the SMA connectors on the Front Panel of the unit.

Markers are 5 volt fixed amplitude signals. Into 50 ohm loads they provide 2.5V signals to trigger TTL or CMOS inputs.

Each Marker has programmable Start (Delay) and Width times. Marker resolution is 32 samples (16 samples in dual channel mode). The maximum marker time is 64K (0-65535) times 32 samples (single channel) or up to 2,097,120 samples.

For each Segment, any or all markers can be Enabled or Disabled. If Markers are Enabled, the Markers are generated only during the first loop of the segment. (If Markers are desired for every loop, use a single segment sequence to generate the markers for each segment occurrence.)

If a Segment is set to Loop Continuously (Repeat Loop enable), the marker enable ON/OFF feature is ignored, and all three markers will be generated, with their previously defined Marker Delay and Width, at the start of each loop of the segment.

Table 13: Marker Signals

Marker Parameter	Value
Marker Delay 1 Marker Delay 2 Marker Delay 3	The Marker Delay for each Marker is used to adjust the start of the Marker output pulse relative to the start of the waveform segment. This allows the Markers to be positioned at any point (within the 65535 register range) in the waveform. If the Marker Delay (1, 2 or 3) is zero, the associated Marker pulse is coincident with start of the segment. The Marker is delayed from the start of the segment by 32 clock periods x N, where N is the value of the delay parameter.
Marker Width 1 Marker Width 2 Marker Width 3	The Marker Width for each Marker is used to adjust the width of the Marker output pulse. If the Marker Width (1, 2 or 3) is zero, the width is 32 samples (16 in Dual Channel Mode). The width will increase by 32 samples (16 in Dual Channel Mode) for every count in this register up to a maximum of 4096 x 32 =131,072 when in Single Channel Mode. (16 x 4096 clocks in Dual Channel Mode)

Marker Parameter	Value
Marker Enable 1 Marker Enable 2 Marker Enable 3	Markers are enabled at run time when a Sequence is started. When a Sequence is run, each marker can be turned: • ON for All Segments, • OFF for All Segments, or • ON or OFF on a Segment by Segment basis (for each Segment in the Sequence being run). If the Segment is set to loop continuously, Marker Enable is ignored and all three markers will appear at the beginning of the segment – each time the segment loops.
Marker Enable	Specifies whether Marker 1, 2 and 3 are turned ON or OFF for segments in the Sequence or whether they are ON for specified segments and OFF for specified segments when branched to. See function an2050_SSetBranchVectorMN for more information.

Marker Example

In the example shown below, when the Sequence is run - Marker 1 is set ON, Marker 2 is set ON, and Marker-3 is set OFF for all segments. The behavior of the markers for each segment is described below and varies depending upon Loop Count, Advance Enable and/or Repeat Enable settings.

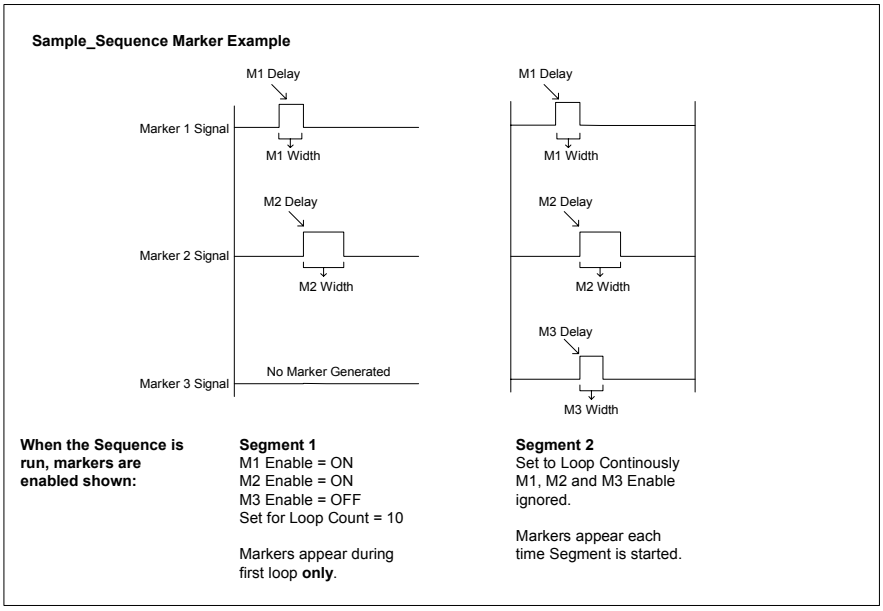


Figure 8: Marker Example

Marker Behavior

Depending whether a Segment is looping continuously, looping according to a **Loop Count**, or in **Advance Mode** waiting for an Advance Trigger, marker behavior may change. The following table lists marker behavior in certain configurations.

Table 14: Marker Behavior in Sequencer Modes

Sequencer Mode	Marker Enable	Description
Loop Count Segment loops number of times specified.	Marker Enable M1, M2, M3 ON/OFF applies	When ON, markers will appear during 1st loop only.
Advance Enable ON Segment loops until Advance Trigger occurs.	Marker Enable M1, M2, M3 ON/OFF applies	When ON, marker will appear during 1st loop only.
Repeat Enable ON Segment loops continuously.	Marker Enable ignored.	All three markers appear each time the segment begins.

4
DBS2050A/2055 Soft Front Panel

Overview

This chapter describes the DBS2050A/2055 Soft Front Panel (SFP), an application designed to familiarize the user with the functionality of the DBS2050A/2055 module. The SFP implements some of the most common driver functions through simple, easy-to-use panels. With the SFP, users can quickly create and generate standard and arbitrary waveform sequences.

This chapter covers:

- Initializing the DBS2050A
- Instruments Panel
- Signal Path
- Controls Panel
- Waveforms Panel
- Segments Panel
- Sequence Panel

The SFP can be used with the DBS2050A instrument as well as with a DBS2055, a 4.8GS/s arbitrary waveform generator. This section describes the use of the SFP with a DBS2050A.

For SFP operation with a DBS2055 refer to the DBS2055 User Manual.

For convenience, the SFP has some unique features layered upon the driver functions, including:

- Autoscaling
- Naming of waveforms, segments and sequences
- Importing of ASCII waveform files
- Quick Run feature, enabling waveforms to be run and previewed prior to being added to a Sequence

The SFP can be used as a programming example for LabWindows/CVI users.

The following sections describe each panel's function. This information is also accessible through the SFP Online Help utility.

Initializing the DBS2050A

The first step in controlling a DBS2050A using the SFP is to find available modules in the chassis and then to select and initialize a module for use.

Communication Protocols

The SFP is capable of communicating with a DBS2050A via two communication methods:

- National Instruments VISA
- Platinum Communication Layer
  which uses a proprietary protocol for communicating with the embedded control computer in the Analogic DP7040/20 chassis.

The SFP makes function calls to the VXIplug&play Driver to communicate with the hardware in the chassis according to the diagram shown below.

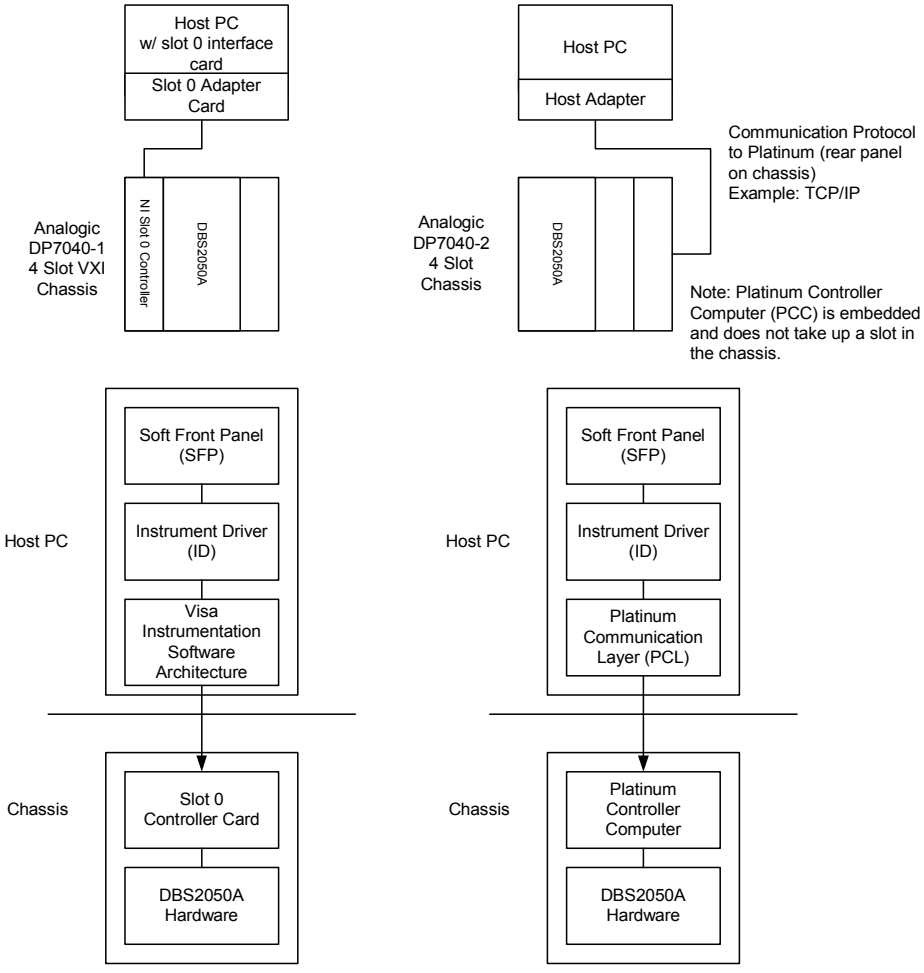


Figure 9: Communication Protocols

Unless an Analogic chassis is being used, the Communication Layer switch on the SFP Initialization Panel should be set to VISA.

Launching the Soft Front Panel Application

Note: If an NI Slot 0 controller is in use, RESMAN must be run prior to launching the SFP. If an Analogic chassis with the Platinum controller computer is in use, it is not necessary to run RESMAN.

To open the SFP for use, select

`Start|Programs|Vxipnp|DBS2050A-2055`

The DBS2050A/2055 Soft Front Panel will open and the Initialization Panel is displayed.

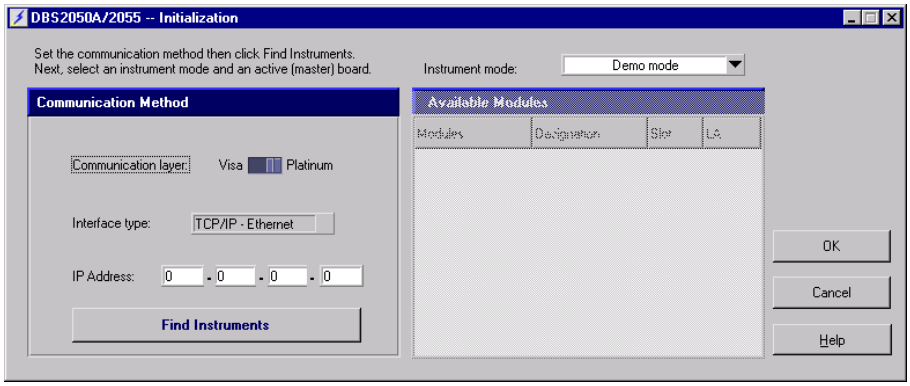
Selecting an Instrument

The Initialization panel is shown below.

- Select *Communication layer* – either Visa or Platinum
- If Platinum is selected, enter the TCP/IP-Ethernet address of the chassis.
- Click *Find Instruments*. The SFP will search the system for all active modules and produce a list in the *Available Modules* window.
- Select *Single DBS2050A* from the Instrument mode drop-down menu or *Demo mode* (to become familiar with SFP functionality without needing to connect to active hardware).

  Note: Synchronous 2050s and 2055 4.8GSa/s modes are only available for use with the DBS2055 instrument. Refer to the DBS2055 User Manual for information regarding these operating modes.
- Highlight the instrument to be initialized from the Available Modules window and click OK.

An EEPROM checksum is checked during module initialization. Checksum failure could be due to either a hardware malfunction or a User Calibration routine that was aborted while storing data into EEPROM. The storing of calibration values is performed at the end of the User Calibration.



Instruments Panel

Once an instrument has been highlighted and initialized from the Available Modules list in the Initialization panel, the Instruments Panel shown below will appear.

Use the Instruments panel to reset or calibrate the active module.

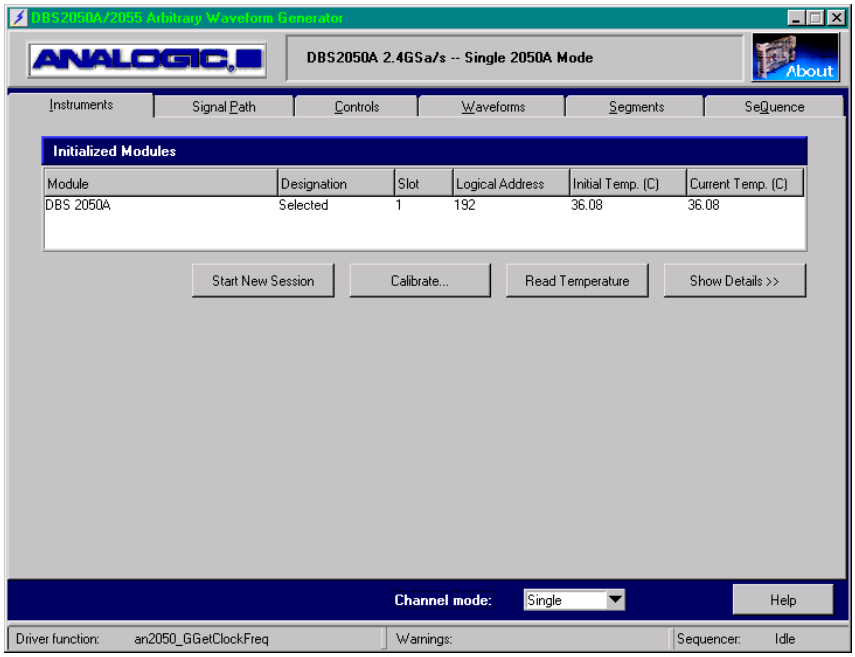


Figure 10: Instruments Panel

Channel Mode

At the bottom of the panel is the Channel mode drop-down menu for channel mode configuration. The DBS2050A signals may be output in Single Channel or Dual Channel mode. Refer to *Chapter 3 Functional Description* for more information on Single and Dual Channel modes.

Single Channel Mode allows an output signal with a sample rate up to 2.4GS/s. Dual Channel Mode allows two independent signals each with a sample rate up to 1.2GS/s.

Note: The selection of channel mode may change the appearance of any of the other tab sheets.

Start New Session

To close the session with the selected module, click the *Start New Session* button. The Initialization panel will re-appear, allowing the User to select another module from the Available Modules list.

Note: Start New Session can only be used if the chassis power has not been reset.

Once an instrument has been highlighted and initialized from the Available Modules list in the Initialization panel, the Instruments Panel shown on the previous page will appear.

User Calibration

A User Calibration software utility is provided to perform user-initiated self-calibration of a DBS2050A instrument.

This utility is called from the SFP when the *Calibrate…* button is pressed on the Instruments panel.

Refer to Chapter 3 for more information about User Calibration.

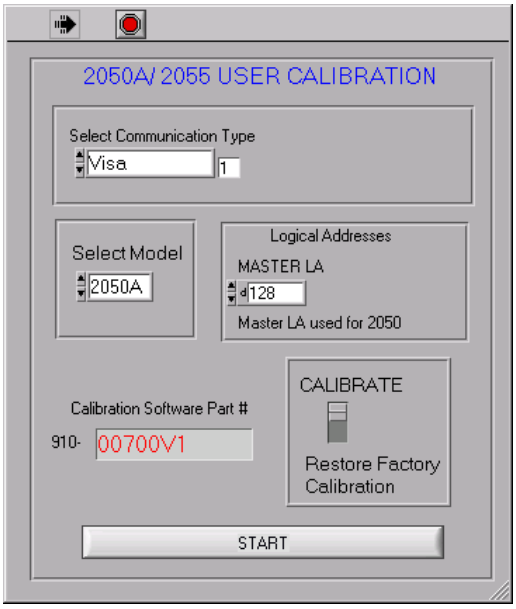
Physical Setup for User Calibration

Performing User Calibration on a DBS2050A is a self-contained software operation and does not require any connections to the DBS2050A.

User Calibration Procedure

To calibrate the module:

- Click the *Calibrate...* button. The 2050A/2055 User Calibration panel will appear.



On the calibration panel, specify the following information.

- Select the Communication Type – either VISA or Platinum (as before on the Initialization panel).
- Select Model – *2050A*.
- Enter the module's Logical Address.
  Enter the logical address in the field labeled MASTER LA.
- Using the switch shown, specify to Restore Factory Calibration settings or to CALIBRATE the unit and generate new calibration values.
- Click *Start*.
- At the end of the calibration, click OK in the "Calibration is Complete" window.

Read Temperature

A current temperature reading of the instrument may be obtained by clicking the *Read Temperature* button. The current reference temperature, within the module, displays in the table.

Show Details

When the *Show Details* button is pressed on the Instruments panel, information is revealed about the DBS205A including serial number and calibration information, see below.

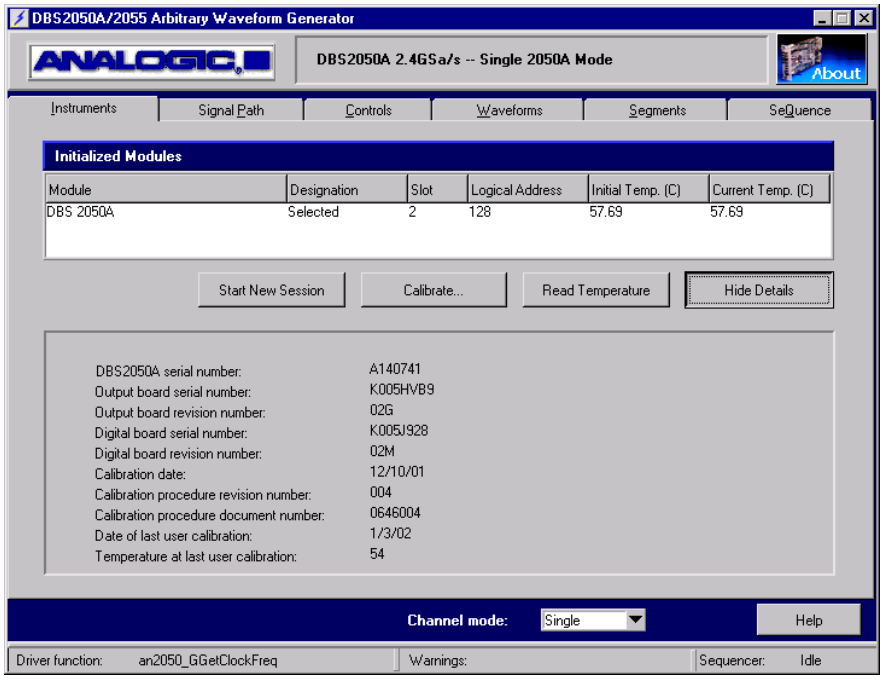


Figure 11: Instruments Panel – Show Details

Signal Path

The Signal Path panel is used to define the configuration of the instrument's output – both Channels are available if Dual Channel mode is selected. Otherwise, only Channel 1 fields will be active for configuration.

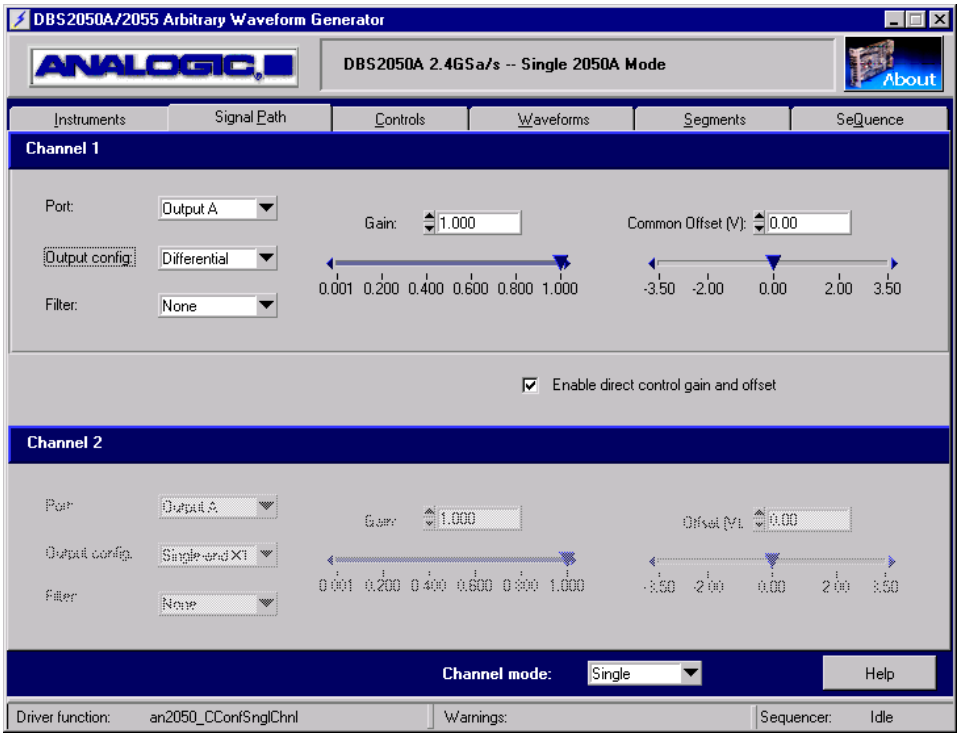


Figure 12: Signal Path Panel

Port (Head)

The *Port* drop-down list allows selection of the SMA connectors on the DBS2050A Front Panel for signal output. Options are:

- Output A (Default)
- Output B

Output config

The *Output Config* drop-down list allows selection of the output configuration for the signal. The options are:

- Differential (Default)
  Not available in Dual Channel Mode
- Single-ended X1
- Single-ended X4

Filter

The DBS2050A available filters are selectable from this drop-down list. The options are:

- None (default)
- 2 MHz Three Pole Low Pass Bessel filter
- 20 MHz Three Pole Low Pass Bessel filter
- 200 MHz Three Pole Low Pass Bessel filter

Direct Gain and Offset Mode

When forming segments in the *Segments* panel, gain and offset values are defined in a Run Time Parameter set and associated with a particular waveform. However, it is possible to override these parameters by setting the Gain and Offset values directly.

Note: Gain and Offset may be changed while a Sequence is running.

The DBS2050A will store user-specified direct gain and offset values until Direct Mode is turned ON. If enabled, the specified values are used for all segments in the Sequence.

- When the *Enable direct control gain and offset* checkbox is checked, the gain and offset values specified in this panel are used.
- When the *Enable direct control gain and offset* checkbox is unchecked, the gain and offset values specified in the *Segments* panel, on a per segment basis, are used.

Direct Gain and Offset Values

Direct Gain and Offset values may be entered using the numeric or slider controls.

The valid range for Gain is 0.001 to 1.000.

The valid range for Offset is –3.50V to +3.50V.

The valid range for Common Mode Offset in Differential output configuration is –3.50V to +3.50V.

Note: Differential offset in Differential output configuration is not supported by the SFP. When programming, however, the VXIplug&play Driver Software allows +2.00V to –2.00V of Differential offset for the Differential output configuration.

Configuring the Output Signals

First, select Single or Dual Channel Mode, if not previously set.

Next, configure the following parameters:

- From the Port drop-down list, select the port to output the waveform.

  Port — *Output A*, *Output B*

- From the Output Configuration drop-down list, select the desired output configuration.

  Output Config — *Differential*, *Single-Ended X1*, *Single-Ended X4*

  Differential produces a signal out of one side of the selected port(1) and the inverse signal out of the other side of that same port(2) – both at X1 gain for a 2Vp-p differential output (for example, 180 degrees out of phase for a sine wave). This option is not available in Dual Channel mode.

  Single-Ended X1 provides maximum bandwidth and an output amplitude up to 1Vp-p.

  Single-Ended X4 provides an output amplitude up to 4Vp-p, but at reduced bandwidth from single-ended X1 configuration.

- From the Filter Select drop-down list, select the desired output low pass filter frequency:

  Filter — None, 2MHz, 20MHz, 200MHz

  None bypasses the output filter, providing maximum bandwidth. *2MHz*, *20MHz and 200MHz* insert low-pass filters in the output signal path.

Controls Panel

The Controls Panel is used to configure

- Sample Clock and Reference Clock
- Main Trigger and Advance Trigger
- Markers

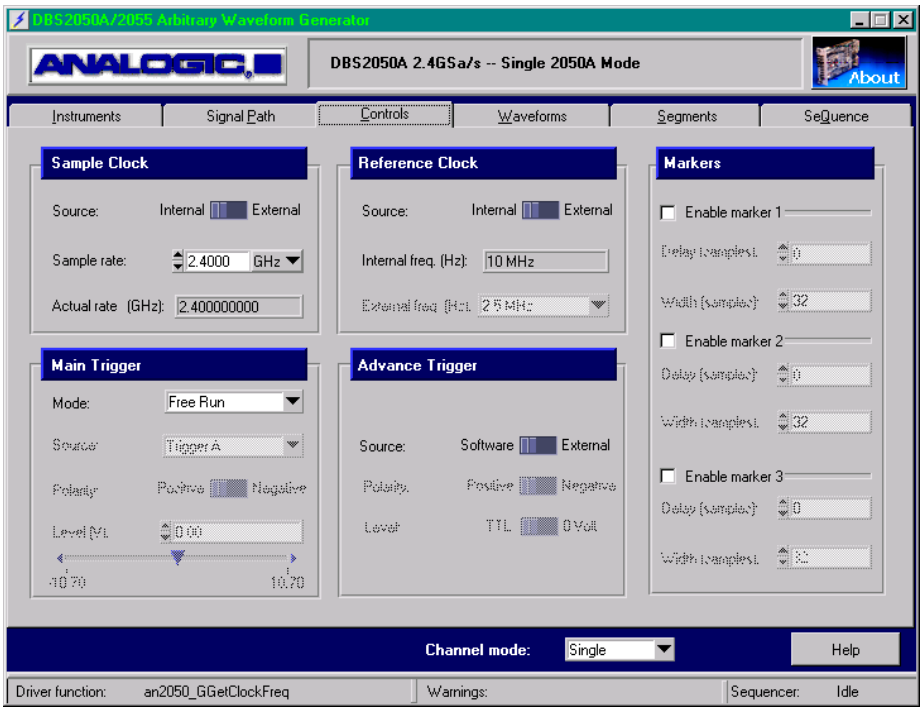


Figure 13: Controls Panel

Sample Clock

Source

Select whether the sample clock to be used is the Internal Sample Clock present in the DBS2050A or an External Sample Clock.

- Internal
  When the Internal Sample Clock is used, the DBS2050A will calculate a sample rate for use as close to the desired sample rate specified as possible.
- External
  When an External clock is specified, the Sample rate specified is assumed to be the External Clock's rate, as well as the desired sampling rate.

Sample Rate

Specify a desired sample rate.

Valid range is 600Hz to 2.4GHz in Single Channel Mode if Internal is selected, 100KHz to 2.4GHz if External is selected.

Valid range is 600Hz to 1.2GHz in Dual Channel Mode.

Reference Clock

If an External Sample Clock is used, the Reference Clock must be internal (this selection is automatic and the Reference Clock source is disabled). If the Internal Sample Clock is selected, either an internal or external Reference Clock can be used.

Source

Select the reference clock source

- Internal
- External

External Freq (Hz)

Specify the frequency of the external reference clock.

The External Reference Clock frequency may be from 2.5MHz to 100MHz in 2.5MHz steps.

Note: The Internal Reference Clock frequency is always 10MHz and is shown as a display-only field on this panel for information purposes.

Main Trigger

Mode

Select the trigger mode. Options are:

- Free Run (Default)
- Start On Trigger
- Stop On Trigger
- Gate by Trigger
- Start/Stop on Trigger

Refer to Chapter 3 for more information regarding these modes.

Source

If not in Free Run mode, specify the source for the main trigger signal. Options are:

- Trigger A SMA input on DBS2050A Front Panel (Default)
- Trigger B
- VXI ECL 0
- VXI ECL 1

Polarity

Select the main trigger polarity, whether to trigger on a positive slope (rising edge) or negative slope (falling edge) of the waveform. Options are:

- Positive
- Negative

Level

Enter the main trigger level to control the trigger threshold voltage, when Trigger A or B is used. A numeric value may be entered or the slider control used to select a voltage.

The range is ±10.0V (0.0 is the Default).

Advance Trigger

Advance triggering will cause the currently running segment to terminate and starts the next segment in the sequence.

Note: The Branch Trigger feature is not available via the Soft Front Panel.

Source

Specify the source for the advance trigger. Options are:

- Software (Default)
  Via the *Advance Trigger* button on the Sequence panel.
- External
  Via the Advance Trigger SMA connector on the DBS2050A Front Panel.

Polarity

If External is selected, the Polarity and Level controls will be active.

Select the polarity from the following options:

- Positive (rising edge)
- Negative (falling edge)

Level

The Advance trigger level options are:

- TTL (Default)
  Predefined voltage set by the factory at 1.5V
- 0 Volts

Markers

Enable Marker Checkbox

Three independent markers are available for use at any time with the DBS2050A. These markers are accessible via SMA connectors on the DBS2050A Front Panel.

Use these checkboxes to select which markers (up to three) will be active when the Sequence is run.

If a segment has looping enabled, markers may be enabled or disabled. If markers are enabled, the markers will be generated, but only at the start of the first loop. If a marker is desired for every loop, using a single segment sequence would cause markers to be generated for each segment occurance.

If a segment has looping disabled, the Enable Marker checkbox is ignored, and all three markers (with the defined Marker Delay and Width) will appear each time the segment begins.

For more information regarding Marker behavior, refer to Chapter 3 Functional Description.

Delay

Specify the Marker Delay (in samples) – the start of the marker.

This value is used to adjust the position of the marker output pulse relative to the start of the waveform segment.

The maximum delay is 2,097,120 samples in Single Channel Mode and 1,048,560 samples in Dual Channel Mode.

For example, if the Marker 1 start position is 0, Marker 1 is coincident with the start of the segment. In Single Channel Mode, the marker delay resolution is 32 clock periods. In Dual Channel Mode, the marker delay resolution is 16 clock periods.

Width

Specify the Marker Width (in samples) – the length of the marker.

The Marker Width adjusts the width of the marker output pulse.

The maximum width is 131,072 samples in Single Channel Mode and 65,536 samples in Dual Channel Mode.

Waveforms Panel

The Waveforms Panel is used to load waveform data to instrument memory. Waveforms may be defined:

- by creating a Standard waveform – sine, square, triangle, noise
- by reading Arbitrary waveform data from a file

Loaded waveforms can be used to create Segments for use in a Sequence.

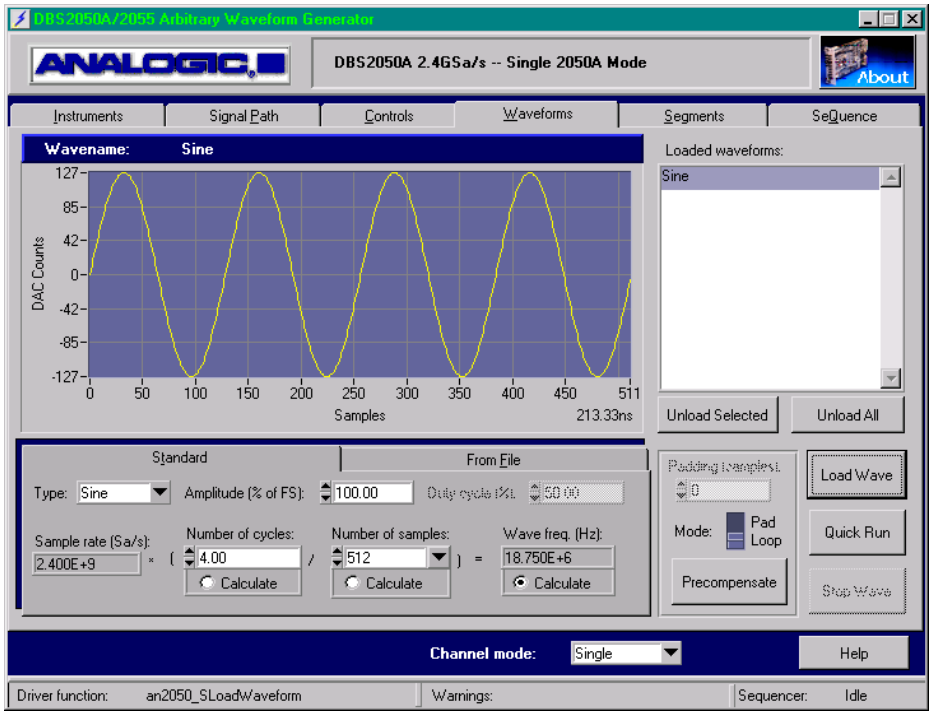


Figure 14: Waveforms Panel

Note about Pre-compensation

Pre-compensation is an optional feature, purchased separately for DBS2050A instruments. Pre-compensation allows creation of higher fidelity waveforms and increases the output bandwidth. For more information, refer to Appendix B, Pre-compensation Option, for hardware configuration and theory of operation.

Creating and Loading a Standard Waveform

To create and load a standard waveform:

1. Define the waveform using the fields in the Standard tab.
2. Pre-compensate the waveform, if desired.
3. Load the waveform using the *Load Wave* button. The waveform appears in the Loaded waveforms table list. The waveform is now loaded into hardware memory and available for use in a Segment.

   The *Unload Selected* and *Unload All* buttons may be used to delete loaded waveforms from memory.
4. To preview the waveform, highlight a loaded waveform and click the *Quick Run* button. A pop-up window will appear – enter a direct gain and offset to be used, if a value other than the defaults (Gain 1 and Offset 0V) is desired, and click OK.

   If the waveform has been pre-compensated, the pre-compensated wave will run when Quick Run is selected.

Changing the name of a loaded waveform

To change the name of a waveform, double-click the waveform name in the *Loaded waveforms* list and re-type the new name, then press the Enter or Return key. Any Segments containing this waveform will be updated accordingly.

Standard tab field descriptions

The Standard tab displayed at the bottom of the Waveforms panel is used to define standard waveforms prior to loading.

Type

Select the standard waveform type from the following list:

- Sine, Square, Triangle and Noise

Amplitude (% of FS)

Select a percentage of the DACs full scale at which the waveform will run.

For maximum accuracy, it is recommended that Amplitude be left at 100% and the gain value adjusted instead using the Gain adjustment when clicking Quick Run or from the segments panel.

Duty Cycle (%)

For Square waves only. Duty cycle establishes the percent in time that a waveform is high with respect to its period (1 cycle).

- Valid range: 0.00 to 100.00.
- Default is 50.0 indicating that the waveform will be 'high' for the same length of time it is 'low'.
- Duty Cycle accuracy is limited by the transitions occurring at the nearest sample times.

Wave Calculator

The Wave Calculator is a collection of four fields on the Standard tab of the Waveforms panel representing:

- Sample rate (Sa/s)
- Number of cycles
- Number of samples
- Wave freq. (Hz)

The Wave Calculator will calculate one of three parameters based on information entered in the other fields. The calculator is a graphical representation of the formula used to calculate wave frequency. The first parameter, Sample Rate, cannot be changed on the Waveforms panel, it must be changed on the Controls panel under the clock sampling rate. Two parameters can be entered to calculate the other parameter. For example, to calculate the number of samples, click on the Calculate button under the Number of Samples control. Enter the desired wave frequency (in Hz) and desired number of cycles. As each value is entered, the number of samples needed will be calculated and displayed. Because the number of samples must be modulo 32 (in Dual Channel Mode, the driver will allow for Modulo 16 waveforms), the calculator may adjust the frequency as needed to obtain valid values. If this happens, and an exact frequency is needed, slowly adjust the number of cycles. The wave frequency will update until the next allowable number of samples is reached. This is the inherent feature of the calculator, if an invalid value is entered into a control, or if a calculated value is out of range, the controls will automatically be coerced to valid values.

Note: When calculating the number of cycles, non-integer values are allowed. It is recommended that once the number of cycles is calculated, the Calculate indicator be changed to calculate the frequency and then the number of cycles be re-entered to the nearest whole number.

Sample Rate

Displays the Sample Rate (in Sa/s) previously entered in the Controls panel.

Number of cycles

Select Calculate or enter the number of waveform cycles desired.

- Valid range: 0.01 to 500,000.00
- Default is 4.00

Number of samples

Select Calculate or enter the total number of waveform samples.

- Valid range is 512 to 8,388,576 (2816x2816)
- Modulo 32
- Default is 512

Wave Frequency

Select Calculate or enter the desired waveform frequency in Hz. If calculated, this frequency is achieved by dividing the Number of cycles by the Number of samples, and then, multiplying this quotient by the Sample Rate.

Loading an Arbitrary Waveform from a File

To load an arbitrary waveform from a file:

1. In the *From File* tab and *Path name* field, enter or browse to the file containing the waveform data to be loaded.
2. Check the Autoscale checkbox, if desired.
3. Pre-compensate the waveform, if desired.
4. Load the waveform using the *Load Wave* button. The waveform will be added to the list in the Loaded waveforms table.

   The *Unload Selected* and *Unload All* buttons may be used to delete loaded waveforms from memory.
5. To preview the waveform, highlight a loaded waveform and click the *Quick Run* button. A pop-up window will appear – enter a direct gain and offset to be used, if a value other than the defaults (Gain 1 and Offset 0V) is desired, and click OK.

   If the waveform has been pre-compensated, the pre-compensated wave will run when Quick Run is selected.

From File tab field descriptions

The From File tab displayed at the bottom of the Waveforms panel is used to import waveform data from an external file.

Path name

The Path name specifies the location of the file. The file name may be entered directly. Alternately, the browse button can be used to locate the file.

Autoscale checkbox

Select Autoscale to scale the waveform data to full scale. The data file is examined and then linearly scaled to the full range of the waveform D/A prior to loading the waveform.

If Autoscale is unchecked, the data is used directly and must be in the range of –127 to +127. Values outside this range will be clipped.

ASCII File Format Requirement

The SFP accepts ASCII text files (.txt file extension) containing waveform data in integer format. The file should be formatted in a single column of integer data values with a NEWLINE (Carriage Return) separating the values.

Segments Panel

The Segments panel is used to associate waveforms with specific gain and offset values, and to define segment behavior.

Once Segments are defined, they can be used in the Sequence panel to create and run a Sequence.

The Ch 2 Available Waveforms fields will remain grayed out unless Dual Channel Mode has been selected.

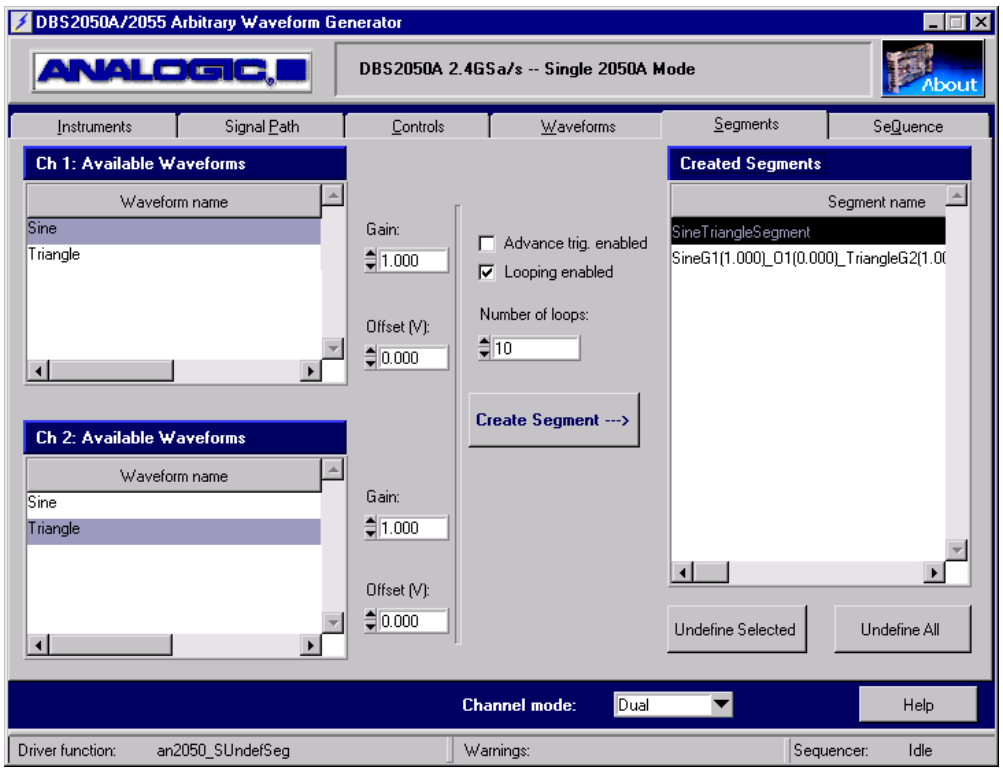


Figure 15: Segments Panel

Creating a Segment

To create a segment:

1. Highlight the desired waveform in the *Waveform name* list in the Ch 1 Available Waveforms table.
2. Set the Gain and Offset values to be applied when this waveform is run.
3. Repeat Steps 1 and 2 for using Ch 2 Available Waveforms if Dual Channel Mode was selected.
4. If the segment is to loop, check the *Looping enabled* checkbox and set the *Number of loops*. If *Looping enabled* is unchecked, all markers will be enabled.

   If the segment is to run continuously until advanced, check the *Advance Trig enabled* checkbox.
5. Click the *Create Segment* button. The segment will be assigned a name and will appear in the *Segment name* list of the Created Segments table. The segment is now available for use in a Sequence.

   Use the *Undefine Selected* and *Undefine All* buttons to remove segments from this list.

Segment Name: Format and How to Change

When a segment is defined, a Segment name is automatically generated. The default segment name is the waveform name appended with the gain and offset for the segment.

The default file name format will be as shown below (dual channel mode segment listed) - listing waveform name, gain and offset information for each channel.

```
SineG1(1.000)_01(0.000)_TriangleG2(1.000)_02(0.000)
```

Scrolling to the right within the *Segment name* window will display more information about the segment, including hardware internal ID numbers assigned to each element of the segment.

To change the assigned name, double-click on the name, type the new segment name and hit Enter to accept.

Segment Panel Field Descriptions

Gain

Gain values may be entered using the numeric control.

The valid range for Gain is 0.001 to 1.000.

Offset

Offset values may be entered using the numeric control.

The valid range for Offset in Single-Ended X1 and Single-Ended X4 output configuration is –3.500V to +3.500V.

Advance Trig enabled

Checkbox to control whether advance triggering will be allowed for this segment.

Looping enabled

This checkbox enables looping and the Number of Loops field.

Note: To avoid a 'trapping' condition when Repeat Loop is disabled, be sure that Advance Trigger is enabled. If they are both disabled, the segment will run indefinitely until the entire Sequence is terminated.

Number of Loops

Valid range is 1 to 4095.

Created Segments

The Created Segments table displays a list of all the Segments that will be available to create a sequence.

The following information is available (scroll to the right to view):

- Segment name
- Gain (CH 1)
- Offset (CH 1)
- ParmID
- Gain (CH 2) – grayed if created in Single Channel Mode
- Offset (CH 2) - grayed if in Single Channel Mode
- ParmID (Slave) - grayed if in Single Channel Mode
- NumLoops
- LoopMode
- WaveID (CH 1) - grayed if in Single Channel Mode
- WaveID (CH 2) - grayed if in Single Channel Mode
- WaveID - grayed if in Single Channel Mode
- WaveID (Slave) - grayed if in Single Channel Mode
- SegID
- SegID (Slave) - grayed if in Single Channel Mode

Sequence Panel

The Sequence panel is used to create and run waveform sequences (made up of previously defined segments).

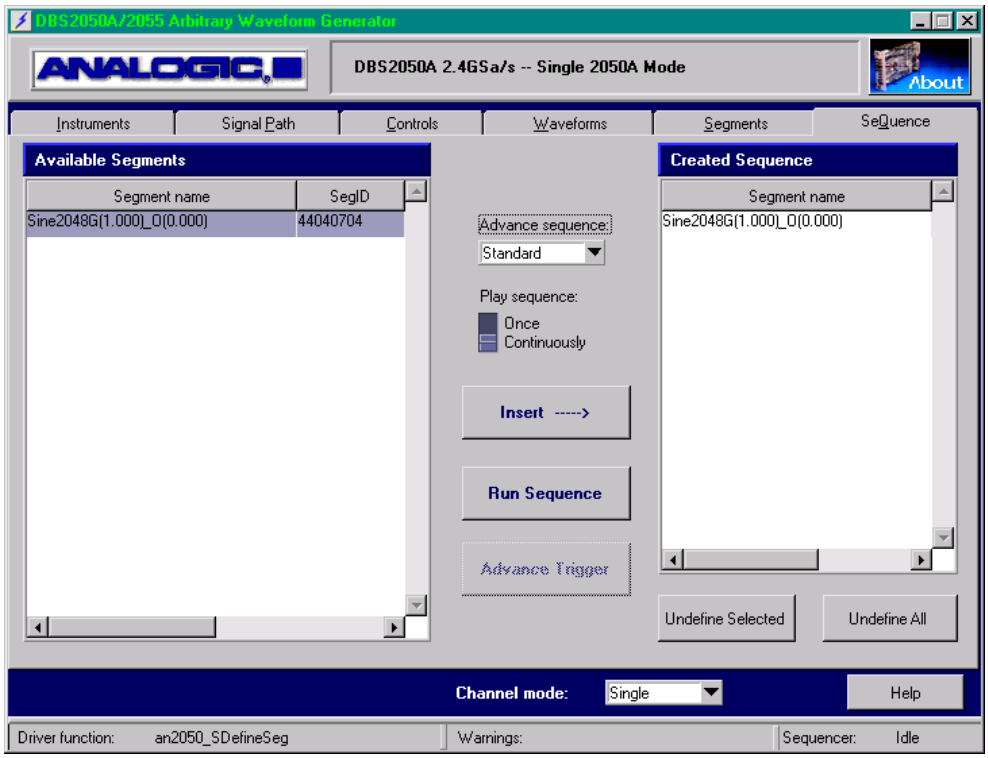


Figure 16: Sequence Panel

Creating and Running a Sequence

To create and run a Sequence:

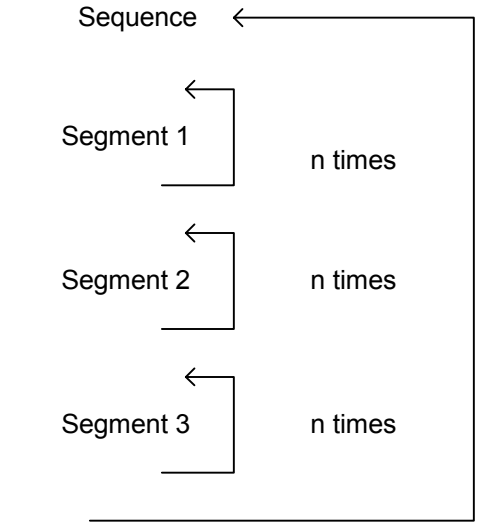
1. Highlight the desired Segment from the Available Segments list.
2. Click *Insert* to add the Segment to the Sequence. Segments can be added to the Sequence multiple times.

   Use the *Undefine Selected* and *Undefine All* buttons to remove segments from Sequence.
3. Repeat Steps 1 and 2 to add additional Segments to the Sequence.
4. If Advance feature is to be used, select whether to Advance via *Standard* (segment will advance at the end of the loop count) or via *On Trigger* (segment will advance when it receives an Advance Trigger, if Advance Trigger was enabled for that segment).

   If *On Trigger* is selected, the Advance Trigger source will be via

the Advance Trigger button on the Sequence panel or the Advance Trigger SMA connector on the DBS2050A Front Panel (depending upon what was configured in the Advance Trigger Source field on the Controls panel).

5. Select whether to play the sequence *Once* or *Continuously*.



Sequence either plays once or continuously

6. When ready, click the *Run Sequence* button to run the Sequence. Click *Stop Sequence* to stop the Sequence.

5
DBS2050A/2055 “VXIplug&play” Driver Software

Introduction

The DBS2050A/2055 VXIplug&play Driver Software provides convenient access to the DBS2050A, and thus relieves the user from having to know the details of DBS2050A hardware. The User can use this driver to create his/her own DBS2050A application. The driver software is released as a .DLL, and supports multi-boards in a chassis. It supports VXI/MXI and IEEE-1394 Firewire™ interfaces.

This driver also serves as the underlying support software for the DBS2050A/2055 Soft Front Panel application that implements some of the most common DBS2050A operations.

The driver provides the capability of maintaining a command/error log. This logging mechanism can be turned on or off, and the logs can be examined through the functions described later in this section.

This driver provides interface functions to make full use of the DBS2050A hardware. All functionality discussed previously in this manual is implemented via driver functions described in this section. The driver also fully implements outputting signals to one channel (Single Channel Output Mode) or to two channels (Dual Channel Output Mode).

The driver keeps 2 sets of clock settings, one used with the internal clock and one with an external clock (if used).

The functions are divided into the following categories:

- Initialize Functions
- Hardware Query
- Shutdown/Reset
- Waveform Creation
- Clock Commands
- Signal Path Control
- Single and Dual Channel Mode Operation
- Unload and Undefine
- Trigger
- Marker
- Informational

CAUTION

The application software that uses this driver should not try to access DBS2050A registers directly. Using functions other than those provided by this driver to access DBS2050A registers may end in unexpected results. However, *reading* register values or getting attribute values using VISA function calls with the Vi Session returned by ‘an2050_init’ is acceptable.

Multi-module Support

This instrument driver supports multiple DBS2050As in the same system. Since 'ViSession' for a given instrument is obtained by calling 'an2050_init()', for efficiency, an application really should not call 'an2050_init()' more than once for any given instrument.

Instrument Control Parameters Maintained by the Driver

The instrument control parameters maintained by the driver are filter, internal clock rate, external clock rate and all clock related items, output configuration, Direct Gain and Offset ON/OFF, and all trigger and marker information.

Standard Waveform Data

The driver supports generation of standard waveforms of sine, square, triangle and random noise. The functions can be implemented so that any number of data points per cycle the user specifies, the instrument driver generates a set of waveform data points that fits both the user's specification and the DBS2050A's waveform restrictions (minimum 512 data points, as well as the number of data points being a multiple of 32).

Waveform Data Memory Management

The DBS2050A has 8 megabytes of waveform data memory. When running a sequence, the DBS2050A requires that waveform data for a given segment come from contiguous memory space. When Waveform Memory becomes fragmented (after a lot of loading and unloading of waveforms), the driver rearranges the Waveform Memory to make room for the new waveform data, and update all the start segment addresses in Segment Memory. This happens automatically when needed, thus relieving the user from having to manage the Waveform Memory and Segment Memory.

Waveform, Segment and Sequence

The DBS2050A supports the defining and execution of a 'Sequence'. A 'Sequence' allows the user to setup a series of waveforms to be run, each with its own specific run parameters (e.g., gain factor, offset, number of loops, start and stop conditions, etc.). Waveform data with a set of Run-Time Parameters form a 'segment', the same waveform with different Run-Time Parameters are considered different 'segments', thus a waveform may be referenced by several 'segments', and unloading of a waveform causes all the 'segments' that reference that waveform to become invalid. Likewise, the 'sequences' that use that waveform also become invalid. A 'sequence' may have up to 4096 'segments'.

The segment data of the waveform to be output must be present in the Segment Memory in order for the Sequencer to output the waveform. The Segment Memory can hold up to 4096 records of segment data. Therefore the total number of anticipated output segments can not exceed 4096 (this includes the segments defined in the normal sequence and the segments defined in all possible branches). There is no restriction on the total number of defined sequences or segments (limited only by the

available host memory). If the host runs out of memory, the user may use 'Undefine' commands to free up the memory occupied by previous definitions. The definitions remain in the host memory until they are explicitly undefined by 'Undefine' commands or implicitly undefined by 'an2050_SUnloadWaveform()' ('unloading' of waveform data makes all segments and sequences using this waveform invalid and the definitions of these segments and sequences are removed from definition list).

Similarly, undefining a run time parameter undefines all segments that use that run time parameter. Likewise, undefining a segment undefines all sequences that use that segment.

The loading of a sequence into Segment Memory occurs automatically when the user issues a function call that needs the segment data records to be in the Segment Memory (e.g., 'an2050_SSetSeq()' or 'an2050_SSetBranchVector()'). The loaded sequence data stays in the Segment Memory until it is explicitly unloaded by the function 'an2050_SUnloadSeq()', or implicitly by functions such as 'an2050_SUndefSeq()', 'an2050_SUndefSeg()' or 'an2050_SUnloadWaveform()', etc.

When unloading waveform data, the entire list of segments is checked to find all the segments that use the waveform data being unloaded. For each segment using the waveform data being unloaded, the entire list of sequences is checked to find and undefine all sequences that use the segment.

Since waveform memory is not accessible when the DBS2050A is running, requests for loading and unloading of the waveform data will not be granted while the Sequencer is running, unless the caller indicates that the Sequencer can be stopped to perform the loading or unloading.

Branching

In addition to normal sequencing, the DBS2050A also supports 'branching.' 'Branching' allows the user to stop the current outputting Sequence and 'branch' to a predefined sequence (and segment within that sequence) when a branch event occurs. Branch vectors are stored in Branch Memory.

Single Channel Mode and Dual Channel Mode

The DBS2050A is designed to support both 'single channel' output mode and 'dual channel' output mode. When in 'dual channel' output mode, the even indexed data is output to channel 1, and the odd indexed data is output to channel 2. In order to support such an arrangement, the software keeps track of whether a waveform data, segment or sequence was defined for single channel or dual channel use. However, the software doesn't prevent the user from using data defined for single channel in dual channel mode, and vice versa.

The software is designed to provide enough functionality to handle the most likely cases without having to load the same waveform data more than once. Functions are provided to allow the user to define waveform

data for dual channel use from loaded waveform data ('an2050_SDefineDualChnlSeg()', 'an2050_SDefineDualChnlSegDup()', 'an2050_SDefineDualChnlSegFromSeg()'). Functions are also provided for directly loading waveform data for dual channel use ('an2050_SLoadDualChnlWave()' functions). If a waveform is to be used for both single channel mode and dual channel mode with this waveform totally output to one channel, the most efficient way to accomplish this is to first load this waveform for single channel use, then use the 'an2050_SDefineDualChnlSeg' function to define waveform data for dual channel use. Note, however, when using the 'an2050_SDefineDualChnlSeg' function to define waveform data for dual channel use from loaded waveform data, the newly created dual channel waveform data is placed in the waveform data memory. Therefore, even though no new waveform data is explicitly loaded by the user, more waveform data memory is used. These commands return 'not enough memory' errors when there is not enough waveform data memory to accommodate the newly created waveform data.

Guidelines for Creating Applications using the DBS2050A VXIplug&play Driver

The following describes how various operations may be achieved with the function set defined in this document.

The first function call in an application should be 'an2050_initInstDrv()' to initialize the DBS2050A Instrument Driver. If 'an2050_initInstDrv()' returns successful, it is safe to proceed to use the instrument.

After the instrument driver is initialized, the DBS2050A must be initialized by calling 'an2050_init()'. To successfully call 'an2050_init()', the logical address of the instrument of interest must be known. If the Logical Address of the instrument is not known, the utility functions 'an2050_find2050Instrs()' or 'an2050_scanChassis()' may be used to obtain the Logical Address.

Output a Waveform with specified run parameter set

Define run parameters using the 'an2050_SDefineRunParms()' function. Then, define a segment associating this run time parameter with the waveform to be output, and define a one-segment sequence using this segment with 'an2050_SDefineSeq()' and run this sequence with 'an2050_SSetSeq()'. It is not necessary to unload the sequence from the Segment Memory after it is done; it may be left in the Segment Memory if future use of this sequence is anticipated.

Output a Sequence with branches set for specific conditions

In addition to the segment and sequence definitions described above, the intended Branch Vectors, Branch Trigger Mode and Branch Vector Source must also be set up (functions: 'an2050_SSetBranchVector()', 'an2050_GBranchVectorSrc()' and 'an2050_GSetBranchTrigMode()') before calling 'an2050_SSetSeq()'. To increase flexibility, the driver does not require an application to go to the beginning of a sequence to access segments in the sequence; instead, the driver allows the

application to ‘branch’ into a desired segment within a sequence. However, once it runs to the end of the sequence, the entire sequence will run again if the sequence loop mode is set to continuous looping.

Functions return an error if the function is not appropriate for the particular machine state, and the user will have to call the function at a later time. For example, a “Sequencer is running” error may be returned when a command that needs access to the Segment Memory is received while the Sequencer is running.

Driver Functions

In addition to the required VXIplug&play functions; ‘an2050_init()’, ‘an2050_close()’, ‘an2050_reset()’, ‘an2050_self_test()’, ‘an2050_revisionQuery()’ and ‘an2050_errorMessage()’, this driver also provides AN2050 specific functions.

All driver functions return a ‘status’. This will be 0 if successful or non-zero for an error or warning.

Initialize Functions

These functions initialize the instrument, set the instrument mode and communication method, and perform a self-test. The following functions are included:

- `an2050_init`
- `an2050_SetInstrumentMode`
- `an2050_ChooseCommMethod`
- `an2050_self_test`

Initialize `an2050_init`

Note: This function has been replaced with an2050_SetInstrumentMode, which should be used to initialize DBS2050A and DBS2055 instruments in all operating modes (Single 2050A, Dual Sync 2050A, 2055 modes). However, the function an2050_init is still included in the driver for those users who have already used it in their applications, or need to conform with the VXIplug&play spec.

This function is used to initialize the DBS2050A instrument by opening the instrument for communication and getting a session handle. It also provides an option to reset the instrument to Power-On Default values. Therefore, ‘resetFlag’ will always reset the board with a valid value of VI_TRUE or VI_FALSE.

When ‘reset,’ the trigger is in Free Run Mode and the clock source is set to internal.

An 'idQuery' is always performed to make sure the unit is DBS2050A and returns 'an2050_ERR_INV_MANF_ID' or 'an2050_ERR_INV_MODEL_CODE' if it is not DBS2050A.

This function checks that the instrument has already connected. If so, an2050_ERR_DEV_IN_USE’ is returned.

The application should not call this function on the same DBS2050A multiple times. If re-initialization of the instrument is desired, use ‘an2050_reset()’.

Default Init States

Control	Default
Sample Clock	Internal
Reference Clock	Internal
Clock Frequency	2.4GHz
Trigger Mode	Free Run Mode
Single/Dual Channel	Single

an2050_init, continued

Function Prototype:

ViStatus **an2050_init** (ViChar _VI_FAR **instrDesc[]**, ViInt16 **idQuery**, Vilnt16 **resetFlag**, ViPSession **vi**)

an2050_init		
Parameters	**Variable Type**	**Description**
<INPUT>		
InstrDesc[]	ViChar_VI_FAR	Specifies the instrument descriptor for the instrument with which a session is to be opened. The format of this field is - "VXI::?::INSTR" where ? is the Logical Address of the instrument. Example 'VXI::7::INSTR' specifies that the instrument has a logical address of 7 and the interface used to connect to it is VXI. Note that for an 'an2050_init()', 'VXI' and 'INSTR' are expected. This 'instrDesc' may be constructed using the following example C code: `sprintf(instrDesc,"VXI::%d::INSTR",LogicAddr);` where 'LogicAddr' is the logical address of the instrument of interest, and may be obtained by using 'an2050_scanChassis()'.
idQuery	ViInt16	Valid Values — Interpretation VI_TRUE — Perform ID Query VI_FALSE — Do not perform ID Query This parameter informs the function to compare the opened instrument's manufacturer's ID and model code with that of the DBS2050A module. If the manufacturer ID does not match, the function returns an2050_ERR_INV_MANF_ID. If the model code does not match, the function returns an2050_ERR_INV_MODEL_CODE. However, for 'an2050' implementation, regardless of the value of 'idQuery', the ID Query is always performed, and the above specified error codes returned if it is not a DBS2050A instrument.
resetFlag	ViInt16	Valid Values — Interpretation VI_TRUE — Perform soft reset VI_FALSE — Do not perform soft reset This parameter informs the function whether or not to perform a soft reset on the instrument.
<OUTPUT>		
vi	ViPSession	This is an output parameter which contains a valid session handle to the instrument which has satisfied the instrument descriptor. If the function fails, this parameter will contain a value of zero. This '*vi' should be used in all driver functions that take a 'ViSession' as an input parameter to access this specific device.
<RETURN>		
		Returns VI_SUCCESS if successful. Possible returns are 'an2050_ERR_INV_MANF_ID' if manufacturer is not 'Analogic', and 'an2050_ERR_INV_MODEL_CODE' if the model is not DBS2050, or returns other errors. Returns 'an2050_ERR_DEV_IN_USE' if the device is in use. Returned error codes may be passed into 'an2050_errorMessage()' for the description of the error.

Set Instrument Mode — `an2050_SetInstrumentMode`

This function should be used to initialize the DBS2050A boards when operating in any of the three valid operating modes:

- Single 2050A mode — DBS2050A, DBS2055
- Dual Synchronous 2050A mode — DBS2055
- 2055 mode — DBS2055

Note: This function replaces the function an2050_init. an2050_init is still supported in the driver for those users who have used it in their programming.

This function provides an option to reset the instrument to Power-On Default values. Therefore, 'resetFlag' will always reset the board with a valid value of VI_TRUE or VI_FALSE. When 'reset', the trigger is in free run mode and the clock source is set to internal.

Function Prototype:

ViStatus **`an2050_SetInstrumentMode`** (ViInt16 **logicAddrMaster**, ViInt16 **logicAddrSlave**, ViInt16 **idQuery**, ViInt16 **resetFlag**, ViInt16 **mode**, ViPSession ***viMaster**, ViPSession ***viSlave**)

`an2050_SetInstrumentMode`		
Parameters	**Variable Type**	**Description**
<INPUT>		
logicAddrMaster	ViInt16	The Logical Address of the Master unit or single 2050A.
logicAddrSlave	ViInt16	The Logical Address of the Slave unit. Enter 0 for Single 2050A mode.
idQuery	ViInt16	This parameter informs the function whether to compare the opened instrument's manufacturer's ID and model code with that of the DBS2050A module. VI_TRUE — Perform ID Query VI_FALSE — Do not perform ID Query
resetFlag	ViInt16	This parameter informs the function whether or not to perform a soft reset on the instrument. VI_TRUE — Perform a soft reset VI_FALSE — Do not perform soft reset
mode	ViInt16	This parameter selects the mode of operation. 0 — DBS2050_SINGLE_MODE — Single DBS2050A mode 1 — DBS2050_SYNC_MODE — DBS2050A Dual Sync mode 2 — DBS2055_MODE — DBS2055 mode
<OUTPUT>		
*viMaster	ViPSession	This parameter contains a valid session handle to the instrument that has satisfied the instrument descriptor. If the function fails, this parameter will contain a value of zero. This '*vi' should be used in all driver functions that take a 'ViSession' as an input parameter to access this device.
*viSlave	ViPSession	This parameter contains a valid session handle to the instrument that has satisfied the instrument descriptor. If the function fails, this parameter will contain a value of zero. This '*vi' should be used in all driver functions that take a 'ViSession' as an input parameter to access this device.

<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. an2050_ERR_INV_MANF_ID if the manufacturer code does not match. an2050_ERR_INV_MODEL_CODE if the model code does not match.

Choose Communication Method `an2050_ChooseCommMethod`

This function is used to define the communication method. Either National Instrument's VISA or Analogic's PCL can be used for communicating with the controller. NI's VISA is the default method.

Users using a standard VXI third party chassis (with a Slot 0 controller) with DBS2050A and DBS2055 instruments should select VISA.

Select PCL if an Analogic DP7040-2 or DP7020-2 chassis is being used. Chassis do not require the use of a Slot 0 controller and allow direct communication to the hardware in the chassis via the PCL communication layer.

Function Prototype:

ViStatus **`an2050_ChooseCommMethod`** (int **comLayerFlag**, char ***physicalMediumString**, char ***chassisAddressString**)

`an2050_ChooseCommMethod`		
Parameters	**Variable Type**	**Description**
<INPUT>		
comLayerFlag	int	This flag toggles between VISA or PCL settings. FLAG_NOT_SET 0 Upon startup the communication flag gets this value PCL_COMMUNICATION 1 Use PCL functionality to communicate to the chassis VISA_COMMUNICATION 2 Use VISA functionality to communicate to the chassis
*physicalMediumString	char	If PCL, this string contains the physical medium for communication with Platinum. If VISA, pass NULL to this parameter.
*chassisAddressString	char	If PCL, this string contains the address of the chassis. If VISA, pass NULL to this parameter.
<OUTPUT>		
None	None	None
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi', other error values. Returned error codes may be passed into 'an2050_errorMessage()' for the description of the error.

Perform a Self Test `an2050_self_test`

This function checks for a valid session handle. It calls 'an2050_invalidSession.'

Function Prototype Format:

ViStatus **`an2050_self_test`** (ViSession **vi**, ViPlnt16 **result**, ViChar _VI_FAR **mesg[]**)

`an2050_self_test`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
*result	ViPInt16 (passed by reference)	ViPInt16 result: 0 = 0K TBC: others to be specified. This parameter returns the value that is passed to it.
mesg	ViChar[]	ViChar _VI_FAR mesg[]: The translated message of the test result.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi', other error values.

Hardware Query

These functions find the number and type of instruments, check the revision and ID numbers, and verify that the session handle is valid. The following functions are included:

- an2050_find2050Instrs
- an2050_scanChassis
- an2050_findVXIInstrs
- an2050_revisionQuery
- an2050_HWRevQuery
- an2050_verifyID
- an2050_VXIInstrID
- an2050_invalidSession

Find All 2050's — an2050_find2050Instrs

This function finds all DBS2050A instruments. See also the description for 'an2050_scanChassis()'.

Function Prototype:

ViStatus **an2050_find2050Instrs** (ViPlnt16 **IaArray**,ViPlnt16 **slotArray**,ViPlnt16 **nDevs**)

an2050_find2050Instrs		
Parameters	**Variable Type**	**Description**
<INPUT>		
nDevs	ViPInt16	As input, 'nDevs' specifies the maximum number of instruments expected. Normally, this is also the array size of 'IaArray' and 'slotArray'. If the total number of instruments found is greater than 'nDevs', only the Logical Addresses and Slot #'s of the first 'nDevs' is output in 'IaArray' and 'slotArray', and the 'nDevs' is updated to reflect the total number of instruments found. Thus, the caller should keep the original input value of 'nDevs' for comparison when this function returns.
<OUTPUT>		
IaArray	ViPInt16[]	Contains the Logical Addresses of all the desired instruments found. The total number of instruments found output in this array is '*nDevs'. The array size of 'IaArray' must be at least the input value of '*nDevs'.
slotArray	ViPInt16[]	Contains the Slot #'s of all the desired instruments found. The total number of instruments found output in this array is '*nDevs'. The array size of 'slotArray' must be at least the input value of '*nDevs'.
*nDevs	ViPInt16 (passed by reference)	As output, '*nDevs' is the total number of desired instruments found. If the total number of desired instruments found is greater than the original input value of '*nDevs', only the Logical Addresses and Slot #s of the first '*nDevs' is output in 'IaArray' and 'slotArray'. The caller should keep the original input value of '*nDevs' for comparison when this function returns.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successfully found the desired instruments. Otherwise, error code that may be passed into 'an2050_errorMessage()' for the description of the error.

Scan VXI Chassis for 2050's `an2050_scanChassis`

This function scans the chassis for all VXI Instruments ((VXIFlag&0xf)=1).

Function Prototype:

ViStatus **`an2050_scanChassis`** (ViInt16 _VI_FAR **IaArray[]**, ViInt16 _VI_FAR **slotArray[]**, ViInt16 ***nDevs**, ViInt16 **VXIFlag**)

`an2050_scanChassis`		
Parameters	**Variable Type**	**Description**
<INPUT>		
nDevs	ViPInt16 (passed by reference)	As input, 'nDevs' specifies the maximum number of instruments expected. Normally, this is also the array size of 'IaArray' and 'slotArray'. If the total number of instruments found is greater than 'nDevs', only the Logical Addresses and Slot #'s of the first 'nDevs' is output in 'IaArray' and 'slotArray', and the 'nDevs' is updated to reflect the total number of instruments found. Thus, the caller should keep the original input value of 'nDevs' for comparison when this function returns.
VXIFlag	ViInt16	bit 0 to bit3 'an2050_SCAN_VXI'(1) to find all VXI instruments 'an2050_SCAN_VXI'(0) to find only DBS2050A instruments bit4 to bit7 0 scan for boards on VXI/MXI interface 1 scan for boards on GPIB-VXI/C interface only all others: scan for boards on both VXI and GPIB-VXI/C interfaces
<OUTPUT>		
IaArray	ViPInt16 []	Contains the Logical Addresses of all the desired instruments found. The total number of instruments found output in this array is 'nDevs'. The array size of 'IaArray' must be at least the input value of 'nDevs'.
slotArray	ViPInt16 []	Contains the Slot #'s of all the desired instruments found. The total number of instruments found output in this array is 'nDevs'. The array size of 'slotArray' must be at least the input value of 'nDevs'.
*nDevs	ViPInt16	As output, 'nDevs' is the total number of desired instruments found. If the total number of desired instruments found is greater than the original input value of 'nDevs', only the Logical Addresses and Slot #s of the first 'nDevs' is output in 'IaArray' and 'slotArray'. The caller should keep the original input value of 'nDevs' for comparison when this function returns.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi', other error values.

Find VXI Instruments `an2050_findVXIInstrs`

This function scans both GPIB-VXI and VXI/MXI interfaces for all VXI instruments present in the chassis.

Function Prototype:

ViStatus **`an2050_findVXIInstrs`** (ViPInt16 **IaArray**, ViPInt16 **slotArray**, ViPInt16 **nDevs**)

Parameters	Variable Type	Description
<INPUT>		
nDevs	ViPInt16 (passed by reference)	As input, 'nDevs' specifies the maximum number of instruments expected. Normally, this is also the array size of 'IaArray' and 'slotArray'. If the total number of instruments found is greater than 'nDevs', only the Logical Addresses and Slot #'s of the first 'nDevs' is output in 'IaArray' and 'slotArray', and the 'nDevs' is updated to reflect the total number of instruments found. Thus, the caller should keep the original input value of 'nDevs' for comparison when this function returns.
<OUTPUT>		
IaArray	ViPInt16 []	Contains the Logical Addresses of all the desired instruments found. The total number of instruments found output in this array is 'nDevs'. The array size of 'IaArray' must be at least the input value of 'nDevs'.
slotArray	ViPInt16 []	Contains the Slot #'s of all the desired instruments found. The total number of instruments found output in this array is 'nDevs'. The array size of 'slotArray' must be at least the input value of 'nDevs'.
nDevs	ViPInt16	As output, 'nDevs' is the total number of desired instruments found. If the total number of desired instruments found is greater than the original input value of 'nDevs', only the Logical Addresses and Slot #s of the first 'nDevs' is output in 'IaArray' and 'slotArray'. The caller should keep the original input value of 'nDevs' for comparison when this function returns.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi', other error values.

Query Revision Number `an2050_revisionQuery`

This function returns the revision number of the instrument driver and the revision number of the firmware.

Function Prototype:

ViStatus **`an2050_revisionQuery`** (ViSession **vi**,ViChar _VI_FAR **driver_rev[]**, ViChar _VI_FAR **instr_rev[]**)

`an2050_revisionQuery`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
driver_rev[]	ViChar _VI_FAR	Buffer to receive revision of the instrument driver. Size of buffer must be at least 20 bytes.
instr_rev[]	ViChar _VI_FAR	Buffer to receive revision of the instrument firmware. Size of buffer must be at least 20 bytes. For DBS2050A, hw_rev. 1, this is "?Not Available?"
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi', other error values.

Hardware Revision Query `an2050_HWRevQuery`

This function returns the hardware revision of the instrument.

Function Prototype:

ViStatus **`an2050_HWRevQuery`** (ViSession **vi**, ViChar _VI_FAR **hw_rev[]**)

`an2050_HWRevQuery`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
hw_rev[]	ViChar _VI_FAR	Buffer to receive the hardware revision of the instrument. Size of buffer must be at least 20 bytes.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi', other error values.

Verify Manufacturer and Model ID `an2050_verifyID`

This function verifies that the manufacturer ID and model code of the opened instrument are the same as those of the DBS2050A.

Function Prototype:

ViStatus **`an2050_verifyID`** (ViSession **theSession**)

`an2050_verifyID`		
Parameters	**Variable Type**	**Description**
<INPUT>		
theSession	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
None	None	None
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi', other error values. Returned error codes may be passed into 'an2050_errorMessage()' for the description of the error.

Output Manufacturer and Model Ids `an2050_VXIInstrID`

This function returns the Manufacturer ID (bit 0 to bit 11 of VXI ID Register) in '*mfrID' and model ID (VXI Device Type Register) in '*modelID' of the VXI instrument specified by 'vi'.

Function Prototype:

ViStatus **`an2050_VXIInstrID`** (ViSession **vi**, ViPUInt16 **mfrID**,ViPUInt16 **modelID**)

`an2050_VXIInstrID`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
mfrID	ViPUInt16	'*mfrID' is the manufacturer ID.
modelID	ViPUInt16	'*modelID' is the value of the Device Type Register.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi', other error values. Returned error codes may be passed into 'an2050_errorMessage()' for the description of the error.

Check if Valid or Invalid Session `an2050_invalidSession`

This function verifies that the input session handle to the instrument is valid. Valid sessions are those output by the 'init()' function. If the session handle is valid, the function returns VI_SUCCESS; if invalid, it returns VI_ERROR_INV_SESSION.

Function Prototype:

ViStatus **`an2050_invalidSession`** (ViSession **vi**)

`an2050_invalidSession`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
None	None	None
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi', other error values.

Shutdown/Reset

These functions reset, test and close the instrument. The following functions are included:

- an2050_reset
- an2050_CResetNTest
- an2050_close
- an2050_closeAll

Reset the 2050 an2050_reset

This function resets the DBS2050A hardware (digital board and output board).

Function Prototype:

ViStatus **an2050_reset** (ViSession **vi**)

an2050_reset		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
None	None	None
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi'. Returned error codes may be passed into 'an2050_errorMessage()' for the description of the error.

Reset and Test `an2050_CResetNTest`

This function performs an 'an2050_reset()' and memory test. This function will stop the Sequencer if it is currently running. All loaded waveforms will be destroyed and all segments and sequences will be undefined as a result of unloading all the waveforms. Existing Run-Time Parameters will be retained. Also, segment memory will be reset to 0.

Function Prototype:

ViStatus **`an2050_CResetNTest`** (ViSession **vi**)

Parameters	Variable Type	Description
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
None	None	None
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi', other error values.

Close 2050 `an2050_close`

This function closes the Vi Session ‘vi’ and frees any memory used by the session with the instrument.

Function Prototype:

ViStatus **`an2050_close`** (ViSession **vi**)

`an2050_close`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
<OUTPUT>		
None	None	None
<RETURN>		
		ViStatus status: ‘VI_SUCCESS’ if successfully closed the session. ‘VI_ERROR_INV_SESSION’, if ‘vi’ is invalid.

Close All 2050As — `an2050_closeAll`

This function closes all VXI instrument sessions opened by the instrument driver and frees the memory used by these sessions.

Note: A DLL may be simultaneously used by many applications. Thus, the ViSessions maintained by this instrument driver may be init'ed by many different applications. "an2050_closeAll()" will close ALL ViSessions, which may include those init'ed by other applications using this driver.

Function Prototype:

ViStatus **`an2050_closeAll`** (ViSession **vi**)

`an2050_closeAll`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
<OUTPUT>		
None	None	None
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successfully closed the session. 'VI_ERROR_INV_SESSION', if 'vi' is invalid.

Waveform Creation

These functions create and load standard and arbitrary waveforms, and calculate the total number of points needed. The following functions are included:

- an2050_CCreateStdWave
- an2050_CCreateStdWaveI
- an2050_CCreateStdWaveT
- an2050_CCrNLoadStdWave
- an2050_CCrNLoadStdWaveI
- an2050_CCrNLoadStdWaveT
- an2050_SLoadWaveform
- an2050_SLoadWaveformN
- an2050_SLoadDualChnlWave2
- an2050_SLoadDualChnlWave2N
- an2050_SLoadDualChnlWaveDup
- an2050_SLoadDualChnlWave
- an2050_SLoadWaveform_2055
- an2050_SLoadWaveformN_2055
- an2050_CCalcTPts
- an2050_CCalcTPtsI

Standard Waveforms

Create Standard Wave

`an2050_CCreateStdWave`

This function calculates the total number of points (‘*tPts’) needed based on the minimum number of points (*nPts) per cycle desired (512 for the DBS2050) and the modulus size (32 for the 2050) required by the hardware, and generates the specified waveform data. The ‘*tPts’ calculated is a multiple of the modulus size and has at least the minimum number of points required by the hardware. The number of points per cycle is close to ‘*nPts’ within a pre-set tolerance (default tolerance is 0.0003, settable using ‘an2050_CSetWaveGenTol()’). If within ‘maxPts’ no total number of points can be found to satisfy the criteria, the total number of points that will give the least deviation from it is output. The ‘*nPts’ will then be re-calculated based on the ‘*tPts’ to be output.

Function Prototype:

ViStatus **`an2050_CCreateStdWave`** (ViSession **vi**, float ***nPts**, ANTUINT32 **maxPts**, float **amp**, float **phase**, short **waveType**, ANTUINT32 ***tPts**, ANTINT8 ****data**)

`an2050_CCreateStdWave`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
*nPts	float	Specifies the number of points per cycle desired. The actual number of points per cycle generated may be slightly different due to the 2050’s restriction of a minimum of 512 points and modulus of 32. The current configuration of standard waveform data generator has the tolerance set at 0.0003, thus the maximum deviation from this function will not be greater than this tolerance.
maxPts	ANTUINT32	Specifies the maximum number of points desired. The actual maximum number of points used in evaluation is the smaller of ‘maxPts’ and size of waveform data memory. Use 0 if the default (size of the DBS2050A waveform data memory) is to be used. Invalid values cause the use of the default value, ‘WAVMaxPts’.
amp	float	Specifies the amplitude of the waveform. Must be between 0 and 1 (full scale).
phase	float	Specifies phase angle of the 1st data point, in radians. Has no effect if ‘waveType’ specified is ‘WAVCWAVENOISE’.
waveType	short	Specifies the type of waveform to generate. Valid values are: WAVCWAVESINE (for sine wave) WAVCWAVESQUARE (for square wave) WAVCWAVETRIANGLE (for triangle wave) WAVCWAVENOISE (for noise) All other values cause ‘WAVEINVPARM’ error return.

<OUTPUT>		
*nPts	float	As output, '*nPts' is the adjusted number of points per cycle.
*tPts	ANTUINT32	The total number of points generated. This will meet the hardware requirement. For 2050, '*tPts' is at least 512 and is a multiple of 32.
**data	ANTINT8	'**data' is the pointer to the waveform data generated. Caller needs to free '**data' when no longer needed.
<RETURN>		
None	None	None

Create Standard Wave (I) `an2050_CCreateStdWaveI`

This function calculates the total number of points (‘*tPts’) needed based on the minimum number of points (*nPts) per cycle desired (512 for the 2050) and the modulus size (32 for the 2050) required by the hardware, and generates the specified waveform data.

The ‘*tPts’ calculated is a multiple of the modulus size and has at least the minimum number of points required by the hardware. The number of points per cycle is close to ‘nPts’. If within ‘maxPts’ no total number of points can be found to satisfy the criteria, the total number of points that gives the least deviation from complete cycles of ‘nPts’ per cycle is output.

Function Prototype:

ViStatus **`an2050_CCreateStdWaveI`** (ViSession **vi**, ANTUINT32 **nPts**, ANTUINT32 **maxPts**, float **amp**, float **phase**, short **waveType**, ANTUINT32 ***tPts**, ANTINT8 ****data**)

`an2050_CCreateStdWaveI`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
nPts	ANTUINT32	Specifies the number of points per cycle desired.
maxPts	ANTUINT32	Specifies the maximum number of points desired. The actual maximum number of points used in evaluation is the smaller of ‘maxPts’ and size of the DBS2050A waveform data memory. Use 0 if the default (size of the DBS2050A waveform data memory) is to be used. Invalid values cause the use of the default value, ‘WAVMaxPts’.
amp	float	Specifies the amplitude of the waveform. Must be between 0 and 1 (full scale).
phase	float	Specifies phase angle of the 1st data point, in radians. This has no effect if ‘waveType’ specified is ‘WAVCWAVENOISE’.
waveType	short	Specifies the type of waveform to generate. Valid values are: WAVCWAVESINE for sine wave) WAVCWAVESQUARE (for square wave) WAVCWAVETRIANGLE (for triangle wave) WAVCWAVENOISE (for noise) All other values cause ‘WAVEINVPARM’ error return.
<OUTPUT>		
*tPts	ANTUINT32	The total number of points generated. This meets the hardware requirement. For 2050, ‘*tPts’ is at least 512 and is a multiple of 32.
**data	ANTINT8	‘**data’ is the pointer to the waveform data generated. Caller needs to free ‘**data’ when no longer needed.
<RETURN>		
None	None	None

Create Standard Wave (T) `an2050_CCreateStdWaveT`

This function uses ‘tPts’ and ‘nPts’ (which can be obtained by calling ‘WAVCalcTPts()’ or ‘WAVCalcTPtsI()’) to generate the specified waveform data. The waveform data generated has ‘nPts’ points per cycle and ‘tPts’ total data points.

Amplitude (‘amp’) should be between 0 and 1 (full scale of ‘dataType’) for integer data type; may be greater than 1 for floating point data types.

Function Prototype:

ViStatus **`an2050_CCreateStdWaveT`** (ViSession **vi**, float **nPts**, float **amp**, float **phase**, short **waveType**, ANTUINT32 **tPts**, ANTINT8 ****data**)

`an2050_CCreateStdWaveT`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
nPts	float	Specifies the number of points per cycle desired.
amp	float	Specifies the amplitude of the waveform. Must be between 0 and 1 (full scale).
phase	float	Specifies phase angle of the 1st data point, in radians. Has no effect if ‘waveType’ specified is ‘WAVCWAVENOISE’.
waveType	short	Specifies the type of waveform to generate. Valid values are: WAVCWAVESINE (for sine wave) WAVCWAVESQUARE (for square wave) WAVCWAVETRIANGLE (for triangle wave) WAVCWAVENOISE (for noise) All other values cause ‘WAVEINVPARM’ error return.
tPts	ANTUINT32	The total number of points generated. This will meet the hardware requirement. For 2050, ‘*tPts’ is at least 512 and is a multiple of 32.
<OUTPUT>		
**data	ANTINT8	‘**data’ is the pointer to the waveform data generated. Caller needs to free ‘**data’ when no longer needed.
<RETURN>		
None	None	None

Create & Load Std. Wave — `an2050_CCrNLoadStdWave`

This function is similar to 'an2050_CCreateStdWave' except it loads the generated standard waveform data into the Waveform Memory for single channel use and assigns a waveform ID, waveID, to the waveform data.

This function calculates the total number of points ('*tPts') needed based on the minimum number of points (*nPts) per cycle desired (512 for the DBS2050) and the modulus size (32 for the 2050) required by the hardware, and generates the specified waveform data. The '*tPts' calculated is a multiple of the modulus size and has at least the minimum number of points required by the hardware. The number of points per cycle is close to '*nPts' within a pre-set tolerance (default tolerance is 0.0003, settable using 'an2050_CSetWaveGenTol()'). If within 'maxPts' no total number of points can be found to satisfy the criteria, the total number of points that will give the least deviation from it is output. The '*nPts' will then be re-calculated based on the '*tPts' to be output.

This function also loads the waveform data into Waveform Memory. A waveform must be loaded into the AN2050 Waveform Memory before it can be used as part of a segment definition. If the Waveform Memory is fragmented, this function will re-arrange if necessary to make room for the waveform data. 'Starting Address' is updated for all segments that are currently loaded into Sequencer Memory. This function takes only waveforms that are multiples of 32, and will return an error if the total number of data points is not a multiple of 32.

Function Prototype:

ViStatus **`an2050_CCrNLoadStdWave`** (ViSession **vi**, float ***nPts**, ANTUINT32 **maxPts**, float **amp**, float **phase**, short **waveType**, ANTUINT32 ***tPts**, ANTID ***waveID**, short **mode**)

`an2050_CCrNLoadStdWave`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
*npts	float	Specifies the number of points per cycle desired. The actual number of points per cycle generated may be slightly different due to the 2050's restriction of a minimum of 512 points and modulus of 32. The current configuration of standard waveform data generator has the tolerance set at 0.0003, thus the maximum deviation from this function will not be greater than this tolerance.
maxPts	ANTUINT32	Specifies the maximum number of points desired. The actual maximum number of points used in evaluation is the smaller of 'maxPts' and the size of the DBS2050A waveform data memory. Use 0 if the default (2050 waveform data memory) is to be used. Invalid values = default value, 'WAVMaxPts'.
amp	float	Specifies the amplitude of the waveform. Must be between 0 and 1 (full scale).
phase	float	Specifies the phase angle of the first data point, in radians. This has no effect if the 'waveType' specified is 'WAVCWAVENOISE'.

waveType	short	Specifies the type of waveform to generate. Valid values are: WAVCWAVESINE (for sine wave) WAVCWAVESQUARE (for square wave) WAVCWAVETRIANGLE (for triangle wave) WAVCWAVENOISE (for noise) All other values cause 'WAVEINVPARM' error return.
mode	short	Specifies action if Sequencer is currently running. 0 Do not load, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and load the waveform.
<OUTPUT>		
*nPts	float	As output, '*nPts' is the adjusted number of points per cycle.
*tPts	ANTUINT32	The total number of points generated. This meets the hardware requirement. For AN2050, '*tPts' is at least 512 and is a multiple of 32.
*waveID	ANTID	'*waveID' is the waveform ID assigned to the generated waveform data.
<RETURN>		
None	None	None

Create & Load Std. Wave (I) `an2050_CCrNLoadStdWaveI`

This function is similar to 'an2050_CCreateStdWaveI' except it loads the generated standard waveform data into the Waveform Memory for single channel use and assigns a waveform ID, waveID, to the waveform data.

This function calculates the total number of points ('*tPts') needed based on the minimum number of points (*nPts) per cycle desired (512 for the DBS2050) and the modulus size (32 for the 2050) required by the hardware, and generates the specified waveform data. The '*tPts' calculated is a multiple of the modulus size and has at least the minimum number of points required by the hardware. The number of points per cycle is close to '*nPts' within a pre-set tolerance (default tolerance is 0.0003, settable using 'an2050_CSetWaveGenTol()'). If within 'maxPts' no total number of points can be found to satisfy the criteria, the total number of points that will give the least deviation from it is output. The '*nPts' will then be re-calculated based on the '*tPts' to be output.

This function loads the waveform data into Waveform Memory. A waveform must be loaded into the AN2050 Waveform Memory before it can be used as part of a segment definition. If the Waveform Memory is fragmented, this function will re-arrange if necessary to make room for the waveform data. 'Starting Address' is updated for all segments that are currently loaded into Sequencer Memory. This function takes only waveforms that are multiples of 32, and will return an error if the total number of data points is not a multiple of 32.

Function Prototype:

ViStatus **`an2050_CCrNLoadStdWaveI`** (ViSession **vi**, ANTUINT32 **nPts**, ANTUINT32 **maxPts**, float **amp**, float **phase**, short **waveType**, ANTUINT32 ***tPts**, ANTID ***waveID**, short **mode**)

`an2050_CCrNLoadStdWaveI`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
nPts	ANTUINT32	Specifies the number of points per cycle desired.
maxPts	ANTUINT32	Specifies the maximum number of points desired. The actual maximum number of points used in evaluation is the smaller of 'maxPts' and the size of the DBS2050A waveform data memory. Use 0 if the default (size of the AN2050 waveform data memory) is to be used. Invalid values cause the use of default value, 'WAVMaxPts'.
amp	float	Specifies the amplitude of the waveform. Must be between 0 and 1 (full scale).
phase	float	Specifies the phase angle of the first data point, in radians. This has no effect if the 'waveType' specified is 'WAVCTYPENOISE'.
waveType	short	Specifies the type of waveform to generate. Valid values are: WAVCWAVESINE (for sine wave) WAVCWAVESQUARE (for square wave)

		WAVCWAVETRIANGLE (for triangle wave) WAVCWAVENOISE (for noise) All other values cause 'WAVEINVPARM' error return.
mode	short	Specifies action if Sequencer is currently running. 0 Do not load, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and load the waveform.
<OUTPUT>		
*tPts	ANTUINT32	The total number of points generated. This meets the hardware requirement. For AN2050, '*tPts' is at least 512 and is a multiple of 32.
*waveID	ANTID	'*waveID' is the waveform ID assigned to the generated waveform data.
<RETURN>		
None	None	None

Create & Load Std. Wave (T) `an2050_CCrNLoadStdWaveT`

This function is similar to 'an2050_CCreateStdWaveT' except it loads the generated standard waveform data into the Waveform Memory for single channel use and assigns a waveform ID, waveID, to the waveform data.

This function calculates the total number of points ('*tPts') needed based on the minimum number of points (*nPts) per cycle desired (512 for the DBS2050) and the modulus size (32 for the 2050) required by the hardware, and generates the specified waveform data. The '*tPts' calculated is a multiple of the modulus size and has at least the minimum number of points required by the hardware. The number of points per cycle is close to '*nPts' within a pre-set tolerance (default tolerance is 0.0003, settable using 'an2050_CSetWaveGenTol()'). If within 'maxPts' no total number of points can be found to satisfy the criteria, the total number of points that will give the least deviation from it is output. The '*nPts' will then be re-calculated based on the '*tPts' to be output.

This function loads the waveform data into Waveform Memory. A waveform must be loaded into the AN2050 Waveform Memory before it can be used as part of a segment definition. If the Waveform Memory is fragmented, this function will re-arrange if necessary to make room for the waveform data. 'Starting Address' is updated for all segments that are currently loaded into Sequencer Memory. This function takes only waveforms that are multiples of 32, and will return an error if the total number of data points is not a multiple of 32.

Function Prototype:

ViStatus **`an2050_CCrNLoadStdWaveT`** (ViSession **vi**, float **nPts**, float **amp**, float **phase**, short **waveType**, ANTUINT32 **tPts**, ANTID ***waveID**, short **mode**)

`an2050_CCrNLoadStdWaveT`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
nPts	float	Specifies the number of points per cycle desired.
amp	float	Specifies the amplitude of the waveform. Must be between 0 and 1 (full scale).
phase	float	Specifies the phase angle of the first data point, in radians. This has no effect if the 'waveType' specified is 'WAVCTYPENOISE'.
waveType	short	Specifies the type of waveform to generate. Valid values are: WAVCWAVESINE (for sine wave) WAVCWAVESQUARE (for square wave) WAVCWAVETRIANGLE (for triangle wave) WAVCWAVENOISE (for noise) All other values cause 'WAVEINVPARM' error return.
mode	short	Specifies action if Sequencer is currently running. 0 Do not load, return an2050_SESQNCRRUNNING

		1 Stop the Sequencer and load the waveform.
tPts	ANTUINT32	The total number of points generated. This, for normal use with the AN2050, should be at least 512 and be a multiple of 32. AWCEINVPARM is returned if 'tPts' does not meet this criteria.
<OUTPUT>		
*waveID	ANTID	'*waveID' is the waveform ID assigned to the generated waveform data.
<RETURN>		
		0 if all OK.

Arbitrary Waveforms

Load Waveform `an2050_SLoadWaveform`

This function loads the waveform data into Waveform Memory. A waveform must be loaded into the AN2050 Waveform Memory before it can be used as part of a segment definition.

If the Waveform Memory is fragmented, this function will re-arrange if necessary to make room for the waveform data. 'Starting Address' is updated for all segments that are currently loaded into Sequencer Memory.

This function takes only waveforms that are multiples of 32, and will return an error if the total number of data points is not a multiple of 32.

Function Prototype:

ViStatus **`an2050_SLoadWaveform`** (ViSession **vi**, ANTINT32 **numPts**, ANTINT8 **data[]**, ANTINT16 **flags**, ANTID ***waveID**, int **mode**)

`an2050_SLoadWaveform`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
numPts	ANTINT32	Specifies the total number of data points in 'data'.
*data	ANTINT8	Contains the waveform data. The array size is 'numPts'. All values in 'data' are interpreted as byte integers.
flags	ANTINT16	Miscellaneous flags. Bit 0 = 0 for Single Channel waveform data. Bit 0 = 1 for Dual Channel waveform data. All other bits must be 0.
mode	int	Specifies action if Sequencer is currently running. 0 Do not load, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and load the waveform.
<OUTPUT>		
*waveID	ANTID	WaveID is a unique number to represent the waveform being loaded.
<RETURN>		
		0 if OK. an2050_MENOMEM: Not enough local memory. an2050_MEVXINOMEM: Not enough An2050 memory to load the waveform data. This means there is not enough total free memory space. To resolve this, the user would need to unload some loaded waveform data in order to make space for this wave. an2050_MENOSEG: All 'an2050_MMaxSegs' segments have been used.

Load Waveform (N) `an2050_SLoadWaveformN`

This function loads the waveform data into Waveform Memory. A waveform must be loaded into the DBS2050A Waveform Memory before it can be used as part of a segment definition. If the Waveform Memory is fragmented, this function will re-arrange if necessary to make room for the waveform data. 'Starting Address' is updated for all segments that are currently loaded into Sequencer Memory.

Function Prototype:

ViStatus **`an2050_SLoadWaveformN`** (ViSession **vi**, ANTINT32 **numPts**, ANTINT8 **data[]**, ANTUINT16 **padMode**, ANTINT32 **maxPts**, ANTINT32 ***tPts**, ANTINT16 **flags**, ANTID ***waveID**, int **mode**)

`an2050_SLoadWaveformN`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
numPts	ANTINT32	Specifies the total number of data points in the pre-existing waveform 'data'.
*data	ANTINT8	Contains the waveform data. The array size is 'numPts'. All values in 'data' are interpreted as byte integers.
padMode	ANTUINT16	Specifies the mode (pad or repeat) and waveform data format. Applies only if 'numPts' is <512 (the minimum number of points for a waveform required by the DBS2050A or is not a multiple of 32 (the modulus size of the DBS2050A). **32768 Repeat Mode (Signed Byte Integer data format)** Instructs the driver to repeat the waveform data until it meets the 512 minimum number of points and the multiple of 32 requirements. If these requirements can't be met on complete cycles within the specified maximum number of points 'maxPts', the driver will pick the best fit that has at least 512 points. The total number of points actually loaded into the Waveform Memory is output in '*tPts'. **33024 Repeat Mode (Offset Binary data format)** Instructs the driver to repeat the waveform data until it meets the 512 minimum number of points and the multiple of 32 requirements. If these requirements can't be met on complete cycles within the specified maximum number of points 'maxPts', the driver will pick the best fit that has at least 512 points. The total number of points actually loaded into the Waveform Memory is output in '*tPts'. **16384 + Pad Value Pad Mode (Signed Byte Integer data format)** Instructs the driver to pad the waveform data with the value specified in bit 0 to bit 7 of 'padMode' until the minimum 512 and multiple of 32 requirements are met. The actual total number of data points loaded after padding is output in '*tPts'. **16384 + Pad Value Pad Mode (Offset Binary data format)** Instructs the driver to pad the waveform data with the value specified in bit 0 to bit 7 of 'padMode' until the minimum 512 and multiple of 32 requirements are met. The actual total number of data points loaded after padding is output in '*tPts'. **0 Instructs driver to return error (AWSEINVPARM) and not to load the data.**

		How to Specify Pad Value **Signed Byte** / **Offset Binary** / **Description** 0x7F (+127) / 0xFF (+127) / maximum positive value 0x01 (+1) / 0x81 (+1) / one LSB above mid value 0x00 (0) / 0x80 (0) / mid value (0 volts) 0x8F (-1) / 0x7F (-1) / one LSB below mid value 0x81 (-127) / 0x01 (-127) / maximum negative value 0x80 (-128) / 0x00 (-128) / one LSB below maximum negative value
maxPts	ANTINT32	If 'numPts' is <512 or not a multiple of 32, this parameter can be used to specify the maximum number of points that will be generated during Repeat Mode. Set this to 0 if the default maximum number of points (currently 8 meg) is to be used.
flags	ANTINT16	Miscellaneous flags. Bit 0 = 0 for Single Channel waveform data. Bit 0 = 1 for Dual Channel waveform data. All other bits must be 0.
mode	int	Specifies action if Sequencer is currently running. 0 Do not load, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and load the waveform.
<OUTPUT>		
*tPts	ANTINT32	The total number of data points actually loaded into waveform data memory.
*waveID	ANTID	WaveID is a unique number to represent the waveform being loaded.
<RETURN>		
		0 if OK. an2050_MENOMEM: Not enough local memory. an2050_MEVXINOMEM: Not enough An2050 memory to load the waveform data. an2050_MENOSEG: All 'an2050_MMaxSegs' segments have been used.

Valid Pad Values (Using Bits 0 to 7)

Signed Byte (Bit8=0)	Offset Binary (Bit8=1)	Description
0x7F (+127)	0xFF (+127)	maximum positive value
0x01 (+1)	0x81 (+1)	one LSB above mid value
0x00 (0)	0x80 (0)	mid value (0 volts)
0x8F (-1)	0x7F (-1)	one LSB below mid value
0x81 (-127)	0x01 (-127)	maximum negative value
0x80 (-128)	0x00 (-128)	one LSB below maximum negative value

Load Dual Channel Wave2 `an2050_SLoadDualChnlWave2`

This function merges two waveform data variables, ‘wave1’ and ‘wave2,’ to form the waveform data for dual channel use, and loads the resulting waveform into Waveform Memory. The waveform data created by the combination of ‘wave1’ and ‘wave2’ is pointed to by the variable ‘waveID’.

Function Prototype:

ViStatus **`an2050_SLoadDualChnlWave2`** (ViSession **vi**, ANTID ***waveID**, ANTINT32 **numPts**, ANTINT8 **wave1**, ANTINT8 **wave2**, int **mode**)

`an2050_SLoadDualChnlWave2`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
numPts	ANTINT32	Specifies the number of data points in each of the waveform data arrays ‘wave1’ and ‘wave2’.
*wave1 *wave2	ANTINT8	Specify the waveform data arrays to be merged to form the dual channel waveform. When running this newly created waveform in dual channel mode, ‘wave1’ data is output to Channel 1 and ‘wave2’ data is output to Channel 2.
mode	int	Specifies action if Sequencer is currently running. 0 Do not load, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and load the waveform.
<OUTPUT>		
*waveID	ANTID	Outputs the wave ID for the newly created dual channel waveform.
<RETURN>		
		0 if OK.

Load Dual Channel Wave2 (N) `an2050_SLoadDualChnlWave2N`

This function merges 2 waveform data, 'wave1' and 'wave2' to form waveform data for dual channel use, and loads the resulting waveform data to waveform data memory. The resulting 'waveID' is the ID for the combined waveform data for dual channel use.

Function Prototype:

ViStatus **`an2050_SLoadDualChnlWave2N`** (ViSession **vi**, ANTID ***waveID**, ANTINT32 **numPts**, ANTUINT16 **padMode1**, ANTUINT16 **padMode2**, ANTINT32 **maxPts**, ANTINT32 ***tPts**, ANTINT8 **wave1**, ANTINT8 **wave2**, int **mode**)

`an2050_SLoadDualChnlWave2N`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
numPts	ANTINT32	Specifies number of data points in each of the waveform data arrays 'wave1' and 'wave2'.
padMode1	ANTUINT16	Bit 8 of this parameter specifies the data format of 'wave1' and the LSB of 'padMode1'. All other bits are in effect only if 'numPts' is smaller than 512 (the minimum number of points for a waveform required by the DBS2050A unit) or is not a multiple of 16 (the modulus size of the DBS2050A for each waveform in dual channel mode). When in effect, the interpretation of this parameter is as follows: Bits 15 and 14 Specify the padding mode of 'wave1'. **Repeat Mode:** Instructs the driver to repeat the waveform data until it meets the 512 minimum number of points and the multiple of 32 requirements. If these requirements can't be met on complete cycles within the specified maximum number of points 'maxPts', the driver will pick the best fit that has at least 512 points. The total number of points actually loaded into the Waveform Memory is output in '*tPts'. **Pad Mode:** Instructs the driver to pad the waveform data with the value specified in bit 0 to bit 7 of 'padMode' until the minimum 512 and multiple of 32 requirements are met. The actual total number of data points loaded after padding is output in '*tPts'. Instructs the driver to return error (AWSEINVPARM) and not to load the data. Bit 8 Specifies the data format in 'data' and the least significant byte of 'padMode'. The format is signed byte integer, i.e., 0x0=0, 0x1=1,…0x7f=+127, 0x80=-128 (not a legitimate value in the 2050), 0x81=-127,…0xff=-1. The format is offset binary format, i.e., 0x0=-128 (not a legitimate value in the 2050), 0x1=-127,…0x7f=-1, 0x80=0,…0xff=+127. Note: This is raw DBS2050A hardware format. Bits 0 to 7 **Pad Value:** Offset binary. This is in effect only if bits 15 and 14 are 01. Data format of this byte depends on bit8 of 'padMode'. When in effect, these bits specify the padding value. If the value of these bits is 0x00 (Bit 8=1) or 0x80 (Bit 8=0), the last value of the waveform data is used to pad. All other bits are not used and should be set to 0 for future compatibility.

an2050_SLoadDualChnlWave2N, continued

padMode2	ANTUINT16	Bit 8 of this parameter specifies the data format of 'wave1' and the LSB of 'padMode2'. All other bits are in effect only if 'numPts' is smaller than 512 (the minimum number of points for a waveform required by the DBS2050A unit) or is not a multiple of 16 (the modulus size of the DBS2050A for each waveform in dual channel mode). When in effect, the interpretation of this parameter is as follows: Bits 15 and 14 Specify the padding mode of 'wave2'. **Repeat Mode:** Instructs the driver to repeat the waveform data until it meets the 512 minimum number of points and the multiple of 32 requirements. If these requirements can't be met on complete cycles within the specified maximum number of points 'maxPts', the driver will pick the best fit that has at least 512 points. The total number of points actually loaded into the Waveform Memory is output in '*tPts'. **Pad Mode:** Instructs the driver to pad the waveform data with the value specified in bit 0 to bit 7 of 'padMode' until the minimum 512 and multiple of 32 requirements are met. The actual total number of data points loaded after padding is output in '*tPts'. Instructs the driver to return error (AWSEINVPARM) and not to load the data. Bit 8 Specifies the data format in 'data' and the least significant byte of 'padMode'. The format is signed byte integer, i.e., 0x0=0, 0x1=1,…0x7f=+127, 0x80=-128 (not a legitimate value in the 2050), 0x81=-127,…0xff=-1. The format is offset binary format, i.e., 0x0=-128 (not a legitimate value in the 2050), 0x1=-127,…0x7f=-1, 0x80=0,…0xff=+127. Note: This is raw DBS2050A hardware format. Bits 0 to 7 **Pad Value:** Offset binary. This is in effect only if bits 15 and 14 are 01. Data format of this byte depends on bit8 of 'padMode'. When in effect, these bits specify the padding value. If the value of these bits is 0x00 (Bit 8=1) or 0x80 (Bit 8=0), the last value of the waveform data is used to pad. All other bits are not used and should be set to 0 for future compatibility.
maxPts	ANTINT32	This is used only if 'padMode's are in effect and bits 15 and 14 of 'padMode's are set to 10 (see above). When in use, this specifies the maximum number of points. Set this to 0 if the default maximum number of points (currently 8 meg) is to be used. This affects both waveforms.
mode	int	Specifies action if Sequencer is currently running. 0 Do not load, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and load the waveform.
<OUTPUT>		
*tPts	ANTINT32	The total number of data points of each waveform actually loaded into waveform data memory.
*waveID	ANTID	WaveID is a unique ID for the newly created dual channel waveform data.
*wave1 *wave2	ANTINT8	The waveform data arrays to be merged to form dual channel waveform data. All values in 'wave1' and 'wave2' are interpreted as byte integers and depend on bit 8 of its 'padMode'. When running this newly created dual channel waveform data in dual channel mode, 'wave1' data will output to Channel 1 and 'wave2' data will output to Channel 2.
<RETURN>		
		0 if OK. an2050_MENOMEM: Not enough local memory. an2050_MEVXINOMEM: Not enough An2050 memory to load the waveform data. an2050_MENOSEG: All 'an2050_MMaxSegs' segments have been used.

Load Dual Channel Wave Dup `an2050_SLoadDualChnlWaveDup`

This function duplicates the waveform data 'wave' for 'dual channel mode' and loads the resulting waveform so that the same wave will output to both Channel 1 and Channel 2.

If a waveform is to be used in both single channel mode *and* dual channel mode, the most efficient way to do this is to load the waveform data as for single channel mode use, then use one of the following to define the segment for dual channel mode use:

- 'an2050_SDefineDualChnlSeg()' if the waveform is to be used with other waveform data to form dual channel mode waveform data
- 'an2050_SDefineDualChnlSegDup()' if this same waveform is to be output to both channels in dual channel mode

Function Prototype:

ViStatus **`an2050_SLoadDualChnlWaveDup`** (ViSession **vi**, ANTID ***waveID**, ANTINT32 **numPts**, ANTINT8 **wave[]**, int **mode**)

`an2050_SLoadDualChnlWaveDup`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
numPts	ANTINT32	Specifies the number of data points in the waveform data array 'wave'.
*wave	ANTINT8	Specifies the waveform data array to be duplicated to form the dual channel waveform data.
mode	int	Specifies action if Sequencer is currently running. 0 Do not load, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and load the waveform.
<OUTPUT>		
*waveID	ANTID	Specifies the waveform ID for the newly created dual channel mode waveform data.
<RETURN>		
		0 if OK.

Load Dual Channel Wave `an2050_SLoadDualChnlWave`

This function can be used to load waveform data that was created for 'dual channel mode.' Odd numbered data is output to Channel 1 and even numbered data is output to Channel 2 to form a continuous waveform.

Function Prototype:

ViStatus **`an2050_SLoadDualChnlWave`** (ViSession **vi**, ANTID ***waveID**, ANTINT32 **numPts**, ANTINT8 **wave[]**, int **mode**)

`an2050_SLoadDualChnlWave`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
numPts	ANTINT32	Specifies the number of data points in the waveform data array 'wave'.
*wave	ANTINT8	Specifies the dual channel waveform data array.
mode	int	Specifies action if Sequencer is currently running. 0 Do not load, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and load the waveform.
<OUTPUT>		
*waveID	ANTID	Specifies the waveform ID for the newly created dual channel mode waveform data.
<RETURN>		
		0 if OK.

Load Waveform for DBS2055 `an2050_SLoadWaveform_2055`

This function loads waveform data to the DBS2055 waveform data memory.

This function assumes the data is in an2050 format. Use the function an2050_SloadWaveformN_2055 to load the waveform data if the data is in regular byte integer format.

Function Prototype:

ViStatus **`an2050_SLoadWaveform_2055`** (ViSession **viMaster**, ViSession **viSlave**, ANTINT32 **numPts**, ANTID ***waveID**, ANTINT8 **data[]**, int **mode**)

`an2050_SLoadWaveform_2055`		
Parameters	**Variable Type**	**Description**
<INPUT>		
viMaster	ViSession	The session handle of the instrument to be acted upon – Master unit. This is obtained by calling 'an2050_SetInstrumentMode'.
viSlave	ViSession	The session handle of the instrument to be acted upon – Slave unit. This is obtained by calling 'an2050_SetInstrumentMode'.
numPts	ANTINT32	The total number of data points in 'data'. This must be at least 1024 and be a multiple of 64.
*data	ANTINT8	The waveform data. Array size is 'numPts'. Note: Single channel has been set internally because the DBS2055 only allows Single Channel waveform data. The data format is an2050 format, i.e., 0x0=-128, 0x1=-127,…0x7f=-1, 0x80=0, …0xff=+127.
mode	Int	Specifies action if Sequencer is currently running. 0 Do not load, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and load the waveform data. The Sequencer remains disabled when this function returns
<OUTPUT>		
*waveID	ANTID	'*waveID' is the waveform ID for the newly loaded waveform data.
<RETURN>		
		ViStatus: VI_SUCCESS if successful, VI_ERROR_INV_SESSION: if 'vi' is invalid, else return error value.

Load Waveform for DBS2055 `an2050_SLoadWaveformN_2055`

This function loads waveform data to the DBS2055 waveform data memory.

This function assumes the data is in an2050 format. Use the function an2050_SloadWaveformN_2055 to load the waveform data if the data is in regular byte integer format.

Function Prototype:

ViStatus **`an2050_SLoadWaveformN_2055`** (ViSession **viMaster**, ViSession **viSlave**, ANTINT32 **numPts**, ANTINT8 **data[]**, ANTUINT16 **padMode,** ANTINT32 **maxPts**, ANTINT32 ***tPts**, ANTID ***waveIDmaster**, ANTID ***waveIDslave**, int **mode**)

`an2050_SLoadWaveformN_2055`		
Parameters	**Variable Type**	**Description**
<INPUT>		
viMaster	ViSession	The session handle of the instrument to be acted upon – Master unit. This is obtained by calling 'an2050_SetInstrumentMode'.
viSlave	ViSession	The session handle of the instrument to be acted upon – Slave unit. This is obtained by calling 'an2050_SetInstrumentMode'.
numPts	ANTINT32	The total number of data points in 'data'. This must be at least 1024 and be a multiple of 64.
*data []	ANTINT8	The waveform data. Array size is 'numPts'. Note: Single channel has been set internally because the DBS2055 only allows Single Channel waveform data. The data format is an2050 format, i.e., 0x0=-128, 0x1=-127,…0x7f=-1, 0x80=0, …0xff=+127.
padMode	ANTUINT16	Specifies how padding should be done. Bit 8 of this parameter specifies the data format of 'data' and the LSB of 'padMode'. All other bits are in effect only if 'numPts' is smaller than 1024 (the minimum number of points for a waveform required by the DBS2055) or is not a multiple of 64 (the modulus size of DBS2055). When in effect, the interpretation is as follows: Bits 15 and 14 specify the padding mode of 'data': 10: Instructs the driver to 'repeat' the waveform data until it meets the 1024 minimum number of points and the multiple of 64 requirements. If these requirements can't be met on complete cycles within the specified maximum number of points 'maxPts', the driver will pick the best fit that has at least 1024 points. And the total number of points that is actually loaded into the waveform data memory will be output in '*tPts'. 01: Instructs the driver to pad the waveform data with the value specified in bit 0 to bit 7 until the minimum 1024 and multiple of 64 requirements are met. The actual total number of data points loaded after padding will be output in '*tPts'. 00: Instructs the driver to return error (AN2050_SEINVPARM) and not to load the data. Bit 8 specifies the data format in 'data' and the least significant byte of 'padMode'. 0: the format is regular byte integer, i.e., 0x0=0, 0x1=1,…0x7f=+127, 0x80=-128, 0x81=-127,…0xff=-1 1: the format is an2050 format, i.e., 0x0=-128, 0x1=-127,…0x7f=-1,

		0x80=0,…0xff=+127. Bits 0 to 7: This is in effect only if bits 15 and 14 are 01. Data format of this byte depends on bit 8 of 'padMode'. When in effect, these bits specify the padding value. If the value of these bits is –128 (if bit 8 of 'padMode' is set, this is 0x00, if that bit is clear, this is 0x80; -128 is not a legitimate value for an2050), the last value of the 'data' data will be use to pad. All other bits are not used and should be set to 0 for future compatibility.
maxPts	ANTINT32	This is used only if 'padMode' is in effect and bits 15 and 14 of 'padMode' is 10. When in use, this parameter specifies the maximum number of points of the waveform. Set this to 0 if the default maximum number of points (currently 8 meg) is to be used.
mode	int	Specifies action if Sequencer is currently running. 0 Do not load, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and load the waveform data. The Sequencer remains disabled when this function returns.
<OUTPUT>		
*tPts	ANTINT32	The total number of data points of the waveform actually loaded into waveform data memory.
*waveIDmaster	ANTID	'*waveIDmaster' is the waveform ID for the newly loaded waveform data of the Master unit.
*waveIDslave	ANTID	'*waveIDslave' is the waveform ID for the newly loaded waveform data of the Slave unit.
<RETURN>		
		ViStatus: VI_SUCCESS if successful, VI_ERROR_INV_SESSION: if 'vi' is invalid, else return error value.

Support

Calculate Total Points `an2050_CCalcTPts`

Given the number of points per cycle desired (‘nPts’), this function calculates the minimum number of points (*tPts) needed based on the hardware properties: ‘WAVtotal’, ‘WAVMaxPts’, ‘WAVMinpts’, ‘WAVmod’. For the 2050, WAVtotal and WAVMaxPts are 8 meg, WAVMinPts is 512 and WAVmod is 32. The minimum number of points is calculated based on the above hardware properties of the waveform data generation context specified by the ‘cntxtID’ that is associated with ‘vi’, and ‘minPts’ so that the ‘*tPts+1’ point will have a phase angle close to zero within ‘WAVtol’, and that the ‘tPts’ is an integer multiple of ‘WAVMod’. If no total number of points can be found to satisfy the ‘WAVtol’ criteria within ‘maxPts,’ the total number of points that gives the smallest deviation from zero is to be output. The ‘nPts’ will then be re-calculated based on the ‘*tPts’ to be output.

Function Prototype:

ViStatus **`an2050_CCalcTPts`** (ViSession **vi**, ANTUINT32 ***tPts**, float ***nPts**, ANTUINT32 **minPts**, ANTUINT32 **maxPts**)

`an2050_CCalcTPts`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
*nPts	float	Specifies the number of points per cycle desired.
minPts	ANTINT32	Specifies the minimum total number of points desired. The actual minimum number of points used in evaluation is the greater of ‘minPts’ and ‘WAVMinPts’. Use 0 if ‘WAVMinPts’ is to be used. Invalid values cause the use of the default value: ‘WAVMinPts’.
maxPts	ANTINT32	Specifies the maximum number of points desired. The actual minimum number of points used in evaluation is the smaller of ‘minPts’ and ‘WAVMaxPts’. Use 0 if ‘WAVMaxPts’ is to be used. Invalid values cause the use of the default value: ‘WAVMaxPts’.
<OUTPUT>		
*tPts	ANTINT32	The total number of points that either satisfies the criteria or is the one that has the least deviation from 0. ‘*tPts’ is always an integer multiple of ‘WAVMod’.
<RETURN>		
None	None	None

Calculate Total Points (I) `an2050_CCalcTPtsI`

Given the number of points per cycle desired, ‘nPts’, this function calculates the minimum number of points needed, ‘*tPts’, based on ‘WAVMaxPts’, ‘WAVMinPts’, ‘WAVMod’ of the waveform generation context specified by the ‘cntxtID’ that is associated with ‘vi’, and ‘minPts’ so that ‘*tPts’ is a multiple of ‘nPts’. If within ‘maxPts’ no total number of points can be found to satisfy these criteria, the total number of points that give the least deviation from complete cycles of ‘nPts’ per cycle is output.

ViStatus **`an2050_CCalcTPtsI`** (ViSession **vi**, ANTUINT32 ***tPts**, float ***nPts**, ANTUINT32 **minPts**, ANTUINT32 **maxPts**)

`an2050_CCalcTPtsI`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
nPts	ANTINT32	Specifies the number of points per cycle desired.
minPts	ANTINT32	Specifies the minimum total number of points desired. The actual minimum number of points used in evaluation is the greater of ‘minPts’ and ‘WAVMinPts’. Use 0 if ‘WAVMinPts’ is to be used. Invalid values cause the use of the default value: ‘WAVMinPts’.
maxPts	ANTINT32	Specifies the maximum number of points desired. The actual maximum number of points used in evaluation is the smaller of ‘minPts’ and ‘WAVMaxPts’. Use 0 if ‘WAVMaxPts’ is to be used. Invalid values cause the use of the default value: ‘WAVMaxPts’.
<OUTPUT>		
*tPts	ANTINT32	The total number of points that either satisfies the criteria or is the one that has the least deviation from 0. ‘*tPts’ is always an integer multiple of ‘WAVMod’.
<RETURN>		
		0 if all OK.

Clock Commands

These functions control internal, external and reference clocks, and low level clock functions. The following functions are included:

- an2050_GSelClock
- an2050_GSetClockFreq
- an2050_GGetClockFreq
- an2050_GGetClockDiv
- an2050_GSetIntClockFreq
- an2050_GSetExtClockFreq
- an2050_GSetExtSrcClockFreq
- an2050_GSelRefClock
- an2050_GSetExtRefFreq
- an2050_GSetGateDelay
- an2050_GSetClockDiv
- an2050_GSetExtClockDiv
- an2050_GSetIntClockDiv

Internal and External Clock Functions

Set Clock Source — an2050_GSelClock

This function is used to select the source clock: internal or external. This function also switches the sampling frequency between the internal clock frequency and the external clock frequency.

Function Prototype:

ViStatus **an2050_GSelClock** (ViSession **vi**, int **sel**)

an2050_GSelClock		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
sel	int	Specifies internal or external clock source. 0 Internal source clock 1 External source clock
<OUTPUT>		
None	None	None
<RETURN>		
None	None	None

Set Currently Selected Clock Frequency `an2050_GSetClockFreq`

This function sets the frequency of the currently selected clock (internal or external). That is, if the currently selected clock is external, this function is equivalent to 'an2050_GSetExtClockFreq' and if the currently selected clock is internal, this function is equivalent to 'an2050_GSetIntClockFreq'.

Function Prototype:

ViStatus **`an2050_GSetClockFreq`** (ViSession **vi**, ANTDOUBLE **freq**)

`an2050_GSetClockFreq`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
freq	ANTDOUBLE	Specifies the clock frequency in MHz.
<OUTPUT>		
None	None	None
<RETURN>		
None	None	None

Get Clock Frequency `an2050_GGetClockFreq`

This function returns the actual sample clock frequency. The actual sample clock frequency is the calculated value that gives the closest approximation to the desired sampling frequency.

When a desired sampling frequency is specified using **an2050_GSetIntClockFreq()** and/or **an2050_GSetClockFreq()**, the internal sample clock frequency is calculated to find the best combination of internal source clock frequency and divider to give the closest approximation for the desired sampling frequency.

Function Prototype:

ViStatus **`an2050_GGetClockFreq`** (ViSession **vi**, int **parm**, ANTDOUBLE ***freq**)

`an2050_GGetClockFreq`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
parm	int	Specifies the Internal or External Clock 0 Internal 1 External
<OUTPUT>		
*freq	ANTDOUBLE	"freq" is the actual frequency of the specified clock.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi', other error values.

Get Clock Divider `an2050_GGetClockDiv`

This function returns the clock divider used when calculating the actual sample clock frequency.

Function Prototype:

ViStatus **`an2050_GGetClockDiv`** (ViSession **vi**, int **parm**, ANTUINT32 ***div**)

`an2050_GGetClockDiv`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
parm	int	Specifies the Internal or External Clock 0 Internal 1 External
<OUTPUT>		
*div	ANTUINT32	"div" is the actual divider used with the specified clock.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi', other error values.

Set Internal Clock Frequency `an2050_GSetIntClockFreq`

Similar to “an2050_GsetExtClockFreq”. Takes effect when internal clock is selected.

This is the only function that sets “an2050_GA”, “an2050_GM” and “an2050_GintDiv”. These variables are used directly without re-calculation when switching from external source clock to internal source clock.

Function Prototype:

ViStatus **`an2050_GsetIntClockFreq`**(ViSession **vi**, ANTDOUBLE **freq**)

`an2050_GsetIntClockFreq`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A..
freq	ANTDOUBLE	Specifies the sampling frequency desired when the internal source clock is selected, in MHz.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. Returns invalid parameter error, an2050_GEINVPARM, if frequency desired is out of range.

Set External Clock Frequency `an2050_GSetExtClockFreq`

This function is used to set the sampling frequency for use with an external source clock.

If the currently selected clock is external, this will take effect immediately, otherwise this frequency is stored until an 'external' clock is selected. When the user switches from internal to external clock, the frequency is set as close as possible to this frequency. Also, a new "external clock divider" kept by the instrument driver is calculated based on the adjusted "freq" and the "external source clock frequency."

*Note: The frequency is set to the nearest possible frequency if the frequency desired is out of range. an2050_GextSrcFreq/(8*64*65538)<freq<an2050_GextSrcFreq. When out of range occurs, this function returns 'an2050_GWINVPARM' if all else is OK, otherwise a VXI bus error or other errors are returned.*

Function Prototype:

ViStatus **`an2050_GSetExtClockFreq`**(ViSession **vi**, ANTDOUBLE **freq**)

`an2050_GSetExtClockFreq`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A..
freq	ANTDOUBLE	Specifies the sampling frequency desired when the external source clock is selected, in MHz.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. Returns invalid parameter error, an2050_GEINVPARM, if frequency desired is out of range.

Set External Source Clock Frequency `an2050_GSetExtSrcClockFreq`

This function specifies the external source clock frequency of the external clock. This changes 'an2050_GExtSrcFreq' and causes 'an2050_GExtClockFreq' to be re-calculated based on 'freq' and 'an2050_GExtDiv'. The divider, 'an2050_GextDiv' will remain the same.

This approach is based on the assumption that when a user changes the external source clock frequency, the intention is to change the sampling frequency associated with the external source clock.

This function must be called to set the external source clock to the correct frequency.

Function Prototype:

ViStatus **`an2050_GSetExtSrcClockFreq`** (ViSession **vi**, ANTDOUBLE **freq**)

`an2050_GSetExtSrcClockFreq`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
freq	ANTDOUBLE	Specifies the external source clock frequency in MHz.
<OUTPUT>		
None	None	None
<RETURN>		
None	None	None

Reference Clock Functions

Select Reference Clock `an2050_GSelRefClock`

This function selects the internal (10MHz) reference clock or an external reference clock.

Note: If the External Reference Clock frequency is not a multiple of 2.5 MHz, then changing the External Reference Clock will result in changing the Internal Clock.

Function Prototype:

ViStatus **`an2050_GSelRefClock`** (ViSession **vi**, int **sel**)

`an2050_GSelRefClock`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
sel	int	Specifies the reference clock source. 0 Set reference clock source to Internal. 1 Set reference clock source to External
<OUTPUT>		
None	None	None
<RETURN>		
None	None	None

Set External Reference Clock Frequency `an2050_GSetExtRefFreq`

This function specifies the frequency of the external reference clock. The frequency must be a multiple of 2.5MHz.

This function will calculate the correct ratio (R) to insure that the PLL phase detector frequency is as close to 2.5MHz as possible. The actual external reference clock frequency can be any value from 2.5MHz to 100MHz in increments of 2.5MHz ± 1%. This insures that the VCO's used in the PLL have sufficient adjustment range for all requested sample rate frequencies.

Function Prototype:

ViStatus **`an2050_GSetExtRefFreq`** (ViSession **vi**, ANTDOUBLE **freq**)

`an2050_GSetExtRefFreq`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
freq	ANTDOUBLE	Specifies the external reference clock frequency in MHz.
<OUTPUT>		
None	None	None
<RETURN>		
None	None	None

Low Level Clock Functions

Set Gate Delay `an2050_CSetGateDelay`

This function specifies a gate delay, from 0 to 255.

It can be used when the hardware configuration includes a synchronized External Clock and External Trigger source.

This command allows the user to set relative timing between an External trigger that is synchronized to a user's External Clock. The trigger is used to start and stop waveforms. The delay eliminates clock period uncertainty of starting waveforms due to set up and hold time requirements in the DBS2050A clock Start/Stop circuitry. Adjustment resolution is 10.5ps typ= 1LSB. Care must be taken by the user to insure that the delays for the external trigger and clock signal are equivalent to prevent waveform time slips as External clock is varied.

Function Prototype:

ViStatus **`an2050_CSetGateDelay`** (ViSession **vi**, ANTINT16 **delay**, int **mode**)

`an2050_CSetGateDelay`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
delay	ANTINT16	Specifies the desired delay. Valid range: 0 to 255
<OUTPUT>		
mode	int	Specifies action if the Sequencer is currently running when function is called. 0 Do not set. Return error. 1 Stop the Sequencer (if running), set the delay, then resume the Sequencer.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_INV_SESSION' if invalid 'vi', other error values.

Set Clock Divider `an2050_GSetClockDiv`

This function sets the divider of a clock for an instrument identified by the session handle. That is, if the currently selected clock is external, this is equivalent to 'an2050_GsetExtClockDiv()', and if the currently selected clock is internal, this is equivalent to 'an2050_GsetIntClockDiv()'.

Function Prototype:

ViStatus **`an2050_GSetClockDiv`**(ViSession **vi**, ANTUINT32 **div**)

`an2050_GSetClockDiv`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
div	ANTUINT32	Specifies the divider. Valid values are 1, 2, 4, 8 to 64 in steps of 8, and 64 to 4,194,304 in steps of 64.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if all OK.

Set External Source Clock Divider `an2050_GSetExtClockDiv`

This function is used to set the clock divider for use with an external source clock. If the currently selected clock is external, this will take effect immediately. Otherwise, this value is stored, and when the user switches from an internal to external source clock, the divider will be set to this value.

This function also updates 'an2050_GExtClockFreq'.

Note: Valid dividers are discrete integers and are not continuous. Thus, if "div" is not a valid divider, the driver will adjust to the nearest valid divider where the reciprocal is nearest to 1/div.

Function Prototype:

ViStatus **`an2050_GSetExtClockDiv`** (ViSession **vi**, ANTUINT32 **div**)

`an2050_GSetExtClockDiv`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
div	ANTUINT32	Specifies the clock divider to be used. Valid values are 1, 2, 4, 8 to 64 in increments of 8, and 64 to 4,194,304 in increments of 64.
<OUTPUT>		
None	None	None
<RETURN>		
None	None	None

Set Internal Source Clock Divider `an2050_GSetIntClockDiv`

This function is used to set the clock divider for use with an internal source clock. If the currently selected clock is internal, this will take effect immediately. Otherwise, this value is stored, and when the user switches from an external to internal source clock, the divider will be set to this value.

This function also updates internal clock frequency according to the divider change.

This function will not cause the re-calculation of the internal source clock frequency. Internal source clock frequency is re-calculated only in SetIntClockFreq() and SetClockFreq() to find the best combination of internal source clock frequency and divider to give the closest approximation for the desired sampling frequency.

Note: Valid dividers are discrete integers and are not continuous. Thus, if "div" is not a valid divider, the driver will adjust to the nearest valid divider where the reciprocal is nearest to 1/div.

Function Prototype:

ViStatus **`an2050_GSetIntClockDiv`** (ViSession **vi**, ANTUINT32 **div**)

`an2050_GSetIntClockDiv`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
div	ANTUINT32	Specifies the clock divider to be used. Valid values are 1, 2, 4, 8 to 64 in increments of 8, and 64 to 4,194,304 in increments of 64.
<OUTPUT>		
None	None	None
<RETURN>		
None	None	None

Signal Path Control

These functions control offset, gain, amplitude, filters and output configuration. The following functions are included:

- an2050_CSetDiffOffset
- an2050_CSetWaveGenTol
- an2050_CSetAmpOffsetDirectOnOff
- an2050_CSetAmpDirectVal
- an2050_CSetOffsetDirectVal
- an2050_CSetFilter
- an2050_CConfSnglChnl
- an2050_CConfDualChnl

Set Differential Offset — an2050_CSetDiffOffset

This function sets a differential offset value for use in Single Channel Mode (with Differential output selected).

If currently in Single Channel Mode with Differential output selected, this will take effect immediately. Otherwise, the desired differential offset 'val' value is stored until the output mode is changed.

Function Prototype:

ViStatus **an2050_CSetDiffOffset** (ViSession **vi**, ANTFLOAT **val**)

an2050_CSetDiffOffset		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
val	ANTFLOAT	Specifies the desired differential offset. 12-bit resolution. Valid range: ±2 volts.
<OUTPUT>		
None	None	None
<RETURN>		
		ViStatus status: 0 if OK. AWCEINVPARM if 'val' is out of range.

Set Wave General Tolerance `an2050_CSetWaveGenTol`

This function sets the tolerance used in an2050_CCalcTPts and an2050_CCalcTPtsI.

Function Prototype:

ViStatus **`an2050_CSetWaveGenTol`** (ViSession **vi**, FLOAT **tol**)

`an2050_CSetWaveGenTol`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
tol	FLOAT	Specifies the desired tolerance. Valid range: .0003
<OUTPUT>		
None	None	None
<RETURN>		
		ViStatus status: 0 if OK. AWCEINVPARM if 'val' is out of range.

Set Amplitude & Offset Direct On/Off `an2050_CSetAmpOffsetDirectOnOff`

This function turns the Gain & Offset Direct Mode ON or OFF for both channels.

Function Prototype:

ViStatus **`an2050_CSetAmpOffsetDirectOnOff`** (ViSession **vi**, int **OnOff**, int **mode**)

`an2050_CSetAmpOffsetDirectOnOff`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
OnOff	int	Specifies to turn Direct mode ON or OFF 0 Turn OFF 1 Turn ON All others are ignored. If this is turned on, the amplitude set using the function 'an2050_SsetAmpDirectVal()' is used.
mode	int	Specifies action if the Sequencer is currently running when function is called. 0 Do not set. 1 Set even if the Sequencer is running.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if all OK.

Set Amplitude Direct Value `an2050_CSetAmpDirectVal`

If Gain & Offset Direct Mode is ON, the driver will set the value to gain direct register and also set offset register with a correction based on the 'val' parameter. If Gain & Offset Direct Mode is OFF, the driver keeps the value and will perform a gain correction when the direct mode is turned ON.

Function Prototype:

ViStatus **`an2050_CSetAmpDirectVal`** (ViSession **vi**, ANTINT16 **chnl**, ANTFLOAT **val**, int **mode**)

`an2050_CSetAmpDirectVal`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
chnl	ANTINT16	Specifies channel: 0 Channel 1 1 Channel 2 All others are ignored.
val	ANTFLOAT	Specifies the gain desired. Valid range is 0 to –63 dB. The gain has a 12 bit resolution. The software rounds off to the nearest achievable gain. All other values cause the gain to be set to the closest end.
mode	int	Specifies action if the Sequencer is currently running when function is called. 0 Do not set. 1 Stop the Sequencer, set, and then resume if running.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if all OK.

Note: for full scale (255 counts) DAC waveform, output amplitude=1Vp-p when gain=0dB in X1 output mode and 4Vp-p in X4 output mode.

Set Offset Direct Value `an2050_CSetOffsetDirectVal`

If Gain & Offset Direct Mode is ON, the driver sets the value to offset direct register. If Gain & Offset Direct Mode is OFF, the driver stores the value and performs offset correction when the direct mode is turned ON.

Function Prototype:

ViStatus **`an2050_CSetOffsetDirectVal`** (ViSession **vi**, ANTINT16 **chnl**, ANTFLOAT **val**, int **mode**)

`an2050_CSetOffsetDirectVal`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
chnl	ANTINT16	Specifies channel: 0 Channel 1 1 Channel 2 All others are ignored.
val	ANTFLOAT	Specifies the offset desired. Valid range is –3.5V to +3.5V. All others cause the offset to be set to the closest end. 12-bit resolution
mode	int	Specifies action if the Sequencer is currently running when function is called. 0 Do not set. 1 Stop the Sequencer, set, and then resume if running.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if all OK.

Set Filter `an2050_CSetFilter`

This function allows for setting either 'No Filter' or any one of the three Low Pass Bessel filters, for a specified channel on the instrument.

Function Prototype:

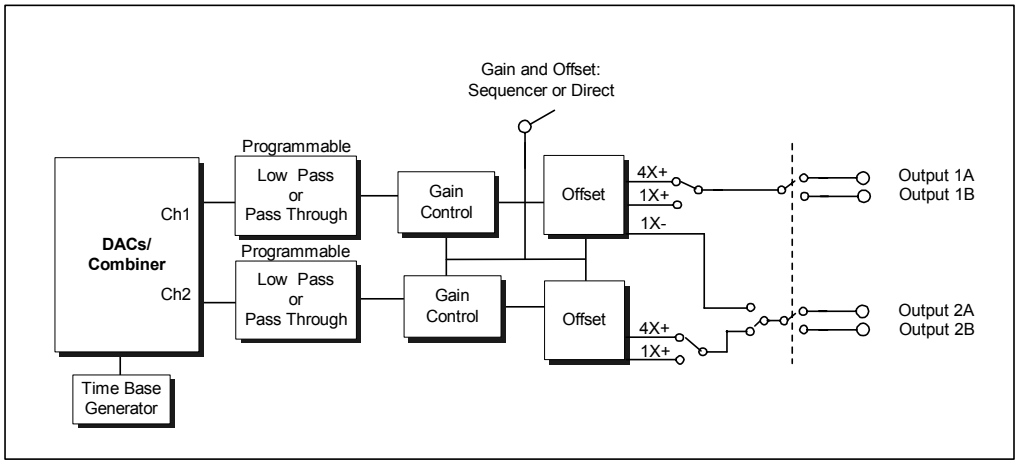
ViStatus **`an2050_CSetFilter`**(ViSession **vi**, int **chnl**, ANTINT16 **filt**, int **mode**)

`an2050_CSetFilter`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the DBS2050A for which the filter is to be set.
chnl	int	Specifies for which channel(s) to set the specified filter 'filt'. 1 Channel 1 2 Channel 2 3 Both Channels
filt	ANTINT16	Specifies the filter to be used. 0 No Filters 1 Use 2 MHz Three Pole Low Pass Bessel filter. 2 Use 20 MHz Three Pole Low Pass Bessel filter. 3 Use 200 MHz Three Pole Low Pass Bessel filter.
mode	int	Specifies action if the Sequencer is currently running when function is called. 0 Do not set. 1 Set even if the Sequencer is running.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK.

Configure Single Channel Output `an2050_CConfSnglChnl`

This function configures the output mode for Single Channel.

The new configuration takes effect immediately if the current output mode is single channel mode. Otherwise, the configuration is stored until the user sets the output mode to single channel mode.



Function Prototype:

ViStatus **`an2050_CConfSnglChnl`** (ViSession **vi**, int **head**, int **load**, int **config**, int **mode**)

`an2050_CConfSnglChnl`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
head	int	Specifies channel 1 output head desired: 0 Head A 1 Head B
load	int	This must be set to: 1 50 ohm load on output.
config	int	Specifies differential output, single-ended X1 output or single-ended X4 output desired: 0 Differential output, positive sense to channel 1and complement to channel 2 of output Head A. 1 Differential output, positive sense to channel 1 and complement to channel 2 of output Head B. 2 Single-ended X1 output. Same as differential mode except there is no channel 2 signal. 3 Single-ended X4 output. Same as single-ended X1 except output is 4x larger.
mode	int	Specifies action if Sequencer is currently running: 0 Do not configure, return an2050_SESQNCRRUNNING. 1 Stop the Sequencer, configure single channel output, then resume Sequencer.
<OUTPUT>		
None	None	None

<RETURN>		
None	None	None

Configure Dual Channel Output `an2050_CConfDualChnl`

This function configures the output mode for Dual Channels.

The new configuration takes effect immediately if the current output mode is dual channel mode. Otherwise, the configuration is stored until the user sets the output mode to dual channel mode.

Function Prototype:

ViStatus **`an2050_CConfDualChnl`** (ViSession **vi**, int **head**, int **load**, int **gain**, int **mode**)

`an2050_CConfDualChnl`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
head	int	Specifies the output heads to be used: 0 Head A for both Channel 1 and Channel 2 1 Head A for Channel 1, Head B for Channel 2 2 Head B for Channel 1, Head A for Channel 2 3 Head B for both Channel 1 and Channel 2
load	int	This parameter must be set to: 3 50 ohm load on both channels
config	int	Specifies no output, single-ended X1 output, or single-ended X4 output. 0 No signal output to either channel: 1 No signal to Channel 1. X1 output to Channel 2. 2 No signal to Channel 1. X4 output to Channel 2. 3 X1 output to Channel 1, no signal to Channel 2. 4 X4 output to Channel 1, no signal to Channel 2. 5 X1 output to both Channel 1 and Channel 2. 6 X1 output for Channel 1, X4 output for Channel 2. 7 X4 output for Channel 1, X1 output for Channel 2. 8 X4 output for both Channel 1 and Channel 2.
mode	int	Specifies action if Sequencer is currently running: 0 Do not configure, return an2050_SESQNCRRUNNING 1 Stop the Sequencer, configure dual channel output, then resume Sequencer.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. AWGEINVPARM if any parameter value is out of range.

Single and Dual Channel Mode Operation

These functions control mode selection, segment, sequence, and Run-Time Parameters. The following functions are included:

- an2050_CSetChnlMode
- an2050_SSetSeqMN
- an2050_SSetSeq
- an2050_SSetSeqM
- an2050_SStopSeq
- an2050_SCopySeq
- an2050_SCopySeq_2055N2050sync
- an2050_SDefineSeg
- an2050_SDefineSeg_2055N2050sync
- an2050_SDefineSeq
- an2050_SDefineSeq_2055N2050sync
- an2050_SDefineSeqP
- an2050_SDefineSeqP_2055N2050sync
- an2050_SDefineDualChnlSeg
- an2050_SDefineDualChnlSeq
- an2050_SDefineDualChnlSegDup
- an2050_SDefineDualChnlSegFromSeg
- an2050_SDefineDualChnlSeg1
- an2050_SSetSeqM_2055N2050sync
- an2050_SSetSeqMN_2055N2050sync
- an2050_SDefineRunParms
- an2050_SDefineRunParms_2055

Single/Dual Channel Mode Select

Set Single/Dual Channel Output Mode `an2050_CSetChnlMode`

This function sets the output mode to single channel mode or dual channel mode. It also calls function an2050_SCorrectAll() to perform gain and offset correction.

Function Prototype:

ViStatus **`an2050_CSetChnlMode`** (ViSession **vi**, ANTINT16 **chnlMode**, int **mode**)

`an2050_CSetChnlMode`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
chnlMode	ANTINT16	Specifies single or dual channel mode: 0 Single Channel Mode 1 Dual Channel Mode
mode	int	Specifies action if the Sequencer is currently running when function is called: 0 Do not set. 1 Stop the Sequencer, set, and then resume if running.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if all OK. an2050_GEINVPARM if “chnlMode” is neither 0 nor 1.

Segment and Sequence Control

Set Sequence (Start Sequencer) with Markers `an2050_SSetSeqMN`

This function loads the specified sequence into the Sequencer Memory (if not already loaded), sets the desired markers as specified by 'm1', 'm2', 'm3', 'm1Array', 'm2Array' and 'm3Array', and enables the running of the sequence.

In normal marker mode, when a Segment has marker(s) turned ON (enabled), the marker pulse appears at 32 clock periods times the associated marker position into the segment, and if the segment is looped in any way, the marker pulse appears in every loop.

If there is not enough Sequencer memory to load the sequence an error is returned.

Function Prototype:

ViStatus **`an2050_SSetSeqMN`** (ViSession **vi**, ANTID **seqID**, int **m1** , int **m2** , int **m3**, char **m1Array[]** , char **m2Array[]**, char **m3Array[]**, ANTID ***seqNew**, int **mode**)

`an2050_SSetSeqMN`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
seqID	ANTID	Specifies the ID of the sequence to be run. An invalid ID causes an error return.
m1	int	0 Turn marker 1 OFF for all segments. 1 Turn marker 1 ON for all segments. 2 Turn marker 1 ON or OFF according to 'm1Array[]'. All other values cause 'AN2050_SEINVPARM' error return.
m2	int	0 Turn marker 2 OFF for all segments. 1 Turn marker 2 ON for all segments. 2 Turn marker 2 ON or OFF according to 'm2Array[]'. All other values cause 'AN2050_SEINVPARM' error return.
m3	int	0 Turn marker 3 OFF for all segments. 1 Turn marker 3 ON for all segments. 2 Turn marker 3 ON or OFF according to 'm3Array[]'. All other values cause 'AN2050_SEINVPARM' error return.
m1Array[]	char	This parameter is used only if 'm1' is 2, otherwise it is ignored. When used, each element specifies whether marker 1 of the corresponding segment (i.e., 'm1Array[0]' specifies the first segment, 'm1Array[1]' specifies the second segment, and so on) should be turned ON (1) or OFF (0). Values other than 0 or 1 cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqID' (may be obtained by calling 'an2050_SGetSeqInfo()').
m2Array[]	char	This parameter is used only if 'm2' is 2, otherwise it is ignored. When used, each element specifies whether marker 2 of the corresponding segment (i.e., 'm2Array[0]' specifies the first segment, 'm2Array[1]' specifies the second segment, and so on) should be turned ON (1) or OFF (0). Values other than 0 or 1 cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqID' (may be obtained by calling 'an2050_SGetSeqInfo()').

m3Array[]	char	This parameter is used only if 'm3' is 2, otherwise it is ignored. When used, each element specifies whether marker 2 of the corresponding segment (i.e., 'm2Array[0]' specifies the first segment, 'm2Array[1]' specifies the second segment, and so on) should be turned ON (1) or OFF (0). Values other than 0 or 1 cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqID' (may be obtained by calling 'an2050_SGetSeqInfo()').
mode	int	Bits 0 to 3 Specifies action if Sequencer is currently running: 0 Do not set, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and set the sequence. Bits 4 to 7 Specifies action if the Sequence is loaded already. Use the loaded sequence, ignore the marker definition specified in 'm1', 'm2', 'm3', 'm1Array', 'm2Array' and 'm3Array'. Use the loaded one but change the marker definition to that specified in the above variables. Keep the loaded one, make a copy of it and load the new one with the new marker definition. If this is specified and 'seqID' is already loaded, the ID of the new copy of 'seqID' is output in '*seqNew'.
<OUTPUT>		
*seqNew	ANTID	'*seqNew' is the ID of the new copy of 'seqID' if 'seqID' is already loaded, and ('mode'&0xf0==0x20). '*seqNew' is 0 if no new copy of the 'seqID' is created (i.e., 'seqID' was not loaded when this function was called). '*seqNew' is meaningful only if 'mode'&0xf0==0x20, otherwise it contains only garbage.
<RETURN>		
		0 if OK. VI_ERROR_INV_SESSION: if 'vi is invalid. an2050_SEINVID: Invalid ID, i.e., 'seqID' is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., 'run sequencer' is enabled). an2050_VERSIONMISMATCH, if hardware is Rev. 0.

Set Sequence (Start Sequencer) `an2050_SSetSeq`

This function loads the specified sequence into the Sequencer Memory (if not already loaded) and enables the running of the sequence. If there is not enough memory to load the sequence, an error is returned.

This function is translated to 'SSetSeqM(vi,SeqID,0,0,0,0,0,(mode|0x10))' to always use the loaded sequence (if already loaded) and set markers off.

Function Prototype:

ViStatus **`an2050_SSetSeq`** (ViSession **vi**, ANTID **seqID**, int **mode**)

`an2050_SSetSeq`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
seqID	ANTID	Specifies the ID of the sequence to be run. An invalid ID causes an error return.
mode	int	Specifies action if Sequencer is currently running: 0 Do not set, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and set the sequence.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., 'seqID' is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., 'run sequencer' is enabled).

Set Sequence (Start Sequencer) with Markers `an2050_SSetSeqM`

Note: This function applies to Rev. 0 Hardware only.

This function loads the specified sequence into the Sequencer Memory (if not already loaded), sets the desired markers as specified by 'm1', 'm2', 'm1Array', and 'm2Array', and enables the running of the sequence. In independent marker mode, when a Segment has marker(s) turned on (enabled), the marker pulse appears at 32 clock periods times the associated marker position (set through the function an2050_GConfigMarkers) into the segment, and if the segment is looped in any way, the marker pulse appears in every loop. If there is not enough memory to load the sequence an error is returned. This function is translated to 'SsetSeqM(vi,seqID,0,0,0,0,0,(mode|0x10))' to always use the loaded sequence (if already loaded) and to always set markers OFF.

Function Prototype:

ViStatus **`an2050_SSetSeqM`** (ViSession **vi**, ANTID **seqID**, int **m1** , int **m2** , char **m1Array[]** , char **m2Array[]**, ANTID ***seqNew**, int **mode**)

`an2050_SSetSeqM`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
seqID	ANTID	Specifies the ID of the sequence to be run. An invalid ID causes an error return.
m1	int	0 Turn marker 1 OFF for all segments. 1 Turn marker 1 ON for all segments. 2 Turn marker 1 ON or OFF according to 'm1Array[]'. All other values cause 'AN2050_SEINVPARM' error return.
m2	int	0 Turn marker 2 OFF for all segments. 1 Turn marker 2 ON for all segments. 2 Turn marker 2 ON or OFF according to 'm2Array[]'. All other values cause 'AN2050_SEINVPARM' error return.
m1Array[]	char	This parameter is used only if 'm1' is 2, otherwise, it is ignored. When used, each element specifies whether marker 1 of the corresponding segment (i.e., 'm1Array[0]' specifies the first segment, 'm1Array[1]' specifies the second segment, and so on) should be turned ON (1) or OFF (0). Values other than 0 or 1 cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqID' (may be obtained by calling 'an2050_SGetSeqInfo()').
m2Array[]	char	This parameter is used only if 'm2' is 2, otherwise, it is ignored. When used, each element specifies whether marker 2 of the corresponding segment (i.e., 'm2Array[0]' specifies the first segment, 'm2Array[1]' specifies the second segment, and so on) should be turned ON (1) or OFF (0). Values other than 0 or 1 cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqID' (may be obtained by calling 'an2050_SGetSeqInfo()').

mode	int	Bits 0 to 3 Specifies action if Sequencer is currently running. 0 Do not set, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and set the sequence. Bits 4 to 7 Specifies action if the Sequence is loaded already. Use the loaded sequence, ignore the marker definition specified in 'm1', 'm2', 'm1Array', and 'm2Array'. Use the loaded one but change the marker definition to that specified in the above variables. Keep the loaded one, make a copy of it and load the new one with the new marker definition. If this is specified and 'seqID' is already loaded, the ID of the new copy of 'seqID' is output in 'seqNew'. The new sequence is loaded.
<OUTPUT>		
*seqNew	ANTID	'*seqNew' is the ID of the new copy of 'seqID' if 'seqID' is already loaded, and ('mode'&0xf0==0x20). '*seqNew' is 0 if no new copy of the 'seqID' is created (i.e., 'seqID' was not loaded when this function was called). '*seqNew' is meaningful only if 'mode'&0xf0==0x20, otherwise it contains only garbage.
<RETURN>		
		0 if OK. VI_ERROR_INV_SESSION: if 'vi is invalid. an2050_SEINVID: Invalid ID, i.e., 'seqID' is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., 'run sequencer' is enabled). an2050_VERSIONMISMATCH, if hardware is Rev. 0.

Stop Sequencer `an2050_SStopSeq`

This function stops the Sequencer if it is currently running. The software clears the "Run Sequencer Enable" bit to stop the Sequencer.

Function Prototype:

ViStatus **`an2050_SStopSeq`** (ViSession **vi**)

`an2050_SStopSeq`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK.

Copy Sequence

`an2050_SCopySeq`

This function copies the sequence, ‘seqID’, to form a new sequence and outputs the ID of the new sequence in ‘*seqNew’. This function does not load the new sequence.

Function Prototype:

ViStatus **`an2050_SCopySeq`** (ViSession **vi**, ANTID **seqID**, ANTID ***seqNew**)

`an2050_SCopySeq`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
seqID	ANTID	Specifies the ID of the sequence to be copied. Invalid ID causes error return and the command is ignored.
<OUTPUT>		
*seqNew	ANTID	‘*seqNew’ is the ID of the new copy of ‘seqID’.
<RETURN>		
		VI_SUCCESS if successful. VI_ERROR_INV_SESSION: if ‘vi’ is invalid. an2050_SEINVID: Invalid ID, i.e., ‘seqID’ is invalid.

Copy Sequence for DBS2055/2050AdualSync `an2050_SCopySeq_2055N2050sync`

This function copies the sequence, 'seqID' (master and slave), to form a new sequence and outputs the ID of the new sequence in '*seqNew' (master and slave). This function does not load the new sequence.

Function Prototype:

ViStatus **`an2050_ScopySeq_2055N2050sync`** (ViSession **viMaster**, ViSession **viSlave**, ANTID **seqIDmaster**, ANTID **seqIDslave**, ANTID ***seqNewMaster**, ANTID ***seqNewSlave**)

`an2050_ScopySeq_2055N2050sync`		
Parameters	**Variable Type**	**Description**
<INPUT>		
viMaster	ViSession	The session handle of the instrument to be acted upon Master unit. This was obtained by calling 'SetInstrumentMode'.
viSlave	ViSession	The session handle of the instrument to be acted upon Slave unit. This was obtained by calling 'SetInstrumentMode'.
seqIDmaster	ANTID	Specifies the Master unit ID of the sequence to be copied. Invalid ID causes error return and the command is ignored.
seqIDslave	ANTID	Specifies the Slave unit ID of the sequence to be copied. Invalid ID causes error return and the command is ignored.
<OUTPUT>		
*seqNewMaster	ANTID	'*seqNewMaster' is the ID of the new copy of 'seqIDmaster'.
seqNewSlave	ANTID	'*seqNewSlave' is the ID of the new copy of 'seqIDslave'.
<RETURN>		
		VI_SUCCESS if successful. VI_ERROR_INV_SESSION: if 'vi' is invalid. an2050_SEINVID: Invalid ID, i.e., 'seqID' is invalid.

Single Channel

Define Segment

`an2050_SDefineSeg`

This function registers a segment definition that is the waveform data identified by 'waveID' and the run parameters identified by 'parmID'. The output 'segID' is used in functions that require 'segmentID' as a parameter. If an 'Out of memory' error occurs, use 'Undefine' commands to free some memory used by obsolete 'parameter', 'segment' or 'sequence' definitions.

Function Prototype:

ViStatus **`an2050_SDefineSeg`** (ViSession **vi**, ANTID **waveID**, ANTID **parmID**, ANTINT16 **loopMode**, ANTID ***segID**)

<table>
<tr><th colspan="3"><code>an2050_SDefineSeg</code></th></tr>
<tr><th>Parameters</th><th>Variable Type</th><th>Description</th></tr>
<tr><td colspan="3"><INPUT></td></tr>
<tr><td>vi</td><td>ViSession</td><td>Specifies the instrument handle for the selected DBS2050A.</td></tr>
<tr><td>waveID</td><td>ANTID</td><td>WaveID is the ID of the waveform data to be associated with the segment being defined.</td></tr>
<tr><td>parmID</td><td>ANTID</td><td>ParmID is the ID of the run parameters to be associated with the segment being defined. 'parmID' may be obtained by a call to 'an2050_SDefineRunParms()'.</td></tr>
<tr><td>loopMode</td><td>ANTINT16</td><td>Variable Type:
Specifies the loop mode and loop count. Set to
<table>
<tr><td>49152 (sets both bits below)</td><td></td></tr>
<tr><td>bit 15: Repeating Loop enable</td><td>If this bit is set, the 'loop count' (bits 0-11) is ignored. The segment repeats until 'Advance to next segment' trigger occurs.
If this bit is clear and bit 14 is clear, the segment repeats the number of times specified in 'loop count'.
If this bit is clear and bit 14 is set, the segment repeats until the advance trigger occurs or the end of loop count is reached, whichever happens first.</td></tr>
<tr><td>bit 14: Advance Trigger enable</td><td>If this bit is clear, 'advance trigger' is ignored.
If this bit is set, see above.
OR</td></tr>
<tr><td>any integer number (loop count)
bit0-bit11:</td><td>Number of loops to be repeated for the segment.
Only valid if bit 15 is clear, otherwise ignored.</td></tr>
</table>
To avoid a 'trapping' condition, make sure that if bit 15 is set, that bit 14 is not set to Ignore Advance Trigger. Otherwise the segment will run indefinitely until the entire Sequence is terminated.</td></tr>
<tr><td colspan="3"><OUTPUT></td></tr>
<tr><td>*segID</td><td>ANTID</td><td>SegID is the ID for the segment just defined.</td></tr>
</table>

<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., either waveID or parmID is invalid. an2050_SENOMEM: Not enough memory. If this happens, the user can use the 'Undef' functions to free up memory occupied by definitions no longer being used.

Define Segment for DBS2055/2050A Dual Sync `an2050_SDefineSeg_2055N2050sync`

This function defines a segment made up of the specified waveform ID and run time parameter ID for the Master and Slave units making up a DBS2055. The output 'segID' (master and slave) is to be used in functions that require 'segmentID' as a parameter. If an 'Out of memory' error occurs, use 'Undefine' commands to free some memory used by obsolete 'parameter', 'segment' or 'sequence' definitions.

Function Prototype:

ViStatus **`an2050_SdefineSeg_2055N2050sync`** (ViSession **viMaster**, ViSession **viSlave**, ANTID **waveIDmaster**, ANTID **waveIDslave**, ANTID **parmIDmaster**, ANTID **parmIDslave**, ANTINT16 **loopMode**, ANTID ***segIDmaster**, ANTID ***segIDslave**)

<table>
<tr><th colspan="3"><code>an2050_SdefineSeg_2055N2050sync</code></th></tr>
<tr><th>Parameters</th><th>Variable Type</th><th>Description</th></tr>
<tr><td colspan="3"><INPUT></td></tr>
<tr><td>viMaster</td><td>ViSession</td><td>The session handle of the instrument to be acted upon Master Unit. This was obtained by calling 'SetInstrumentMode'.</td></tr>
<tr><td>viSlave</td><td>ViSession</td><td>The session handle of the instrument to be acted upon Slave unit. This was obtained by calling 'SetInstrumentMode'.</td></tr>
<tr><td>waveIDmaster</td><td>ANTID</td><td>Specifies the waveform data ID to be associated with the segment being defined on the Master unit.</td></tr>
<tr><td>waveIDslave</td><td>ANTID</td><td>Specifies the waveform data ID to be associated with the segment being defined on the Slave unit.</td></tr>
<tr><td>parmIDmaster</td><td>ANTID</td><td>Specifies the ID of the Run-Time Parameters for the segment being defined in the Master unit.</td></tr>
<tr><td>parmIDslave</td><td>ANTID</td><td>Specifies the ID of the Run-Time Parameters for the segment being defined in the Slave unit.</td></tr>
<tr><td>loopMode</td><td>ANTINT16</td><td>Specifies the loop mode and loop count for the segment. Set to
<table>
<tr><td>49152 (sets both bits below)
bit 15: Repeating Loop enable</td><td>If this bit is set, the 'loop count' (bits 0-11) is ignored. The segment repeats until 'Advance to next segment' trigger occurs.
If this bit is clear and bit 14 is clear, the segment repeats the number of times specified in 'loop count'.
If this bit is clear and bit 14 is set, the segment repeats until the advance trigger occurs or the end of loop count is reached, whichever happens first.</td></tr>
<tr><td>bit 14: Advance Trigger enable</td><td>If this bit is clear, 'advance trigger' is ignored.
If this bit is set, see above.
OR</td></tr>
<tr><td>any integer number (loop count)
bit0-bit11:</td><td>Number of loops to be repeated for the segment.
Only valid if bit 15 is clear, otherwise ignored.</td></tr>
</table>
To avoid a 'trapping' condition, make sure that if bit 15 is set, that bit 14 is not set to Ignore Advance Trigger. Otherwise the segment will run indefinitely until the entire Sequence is terminated.</td></tr>
</table>

<OUTPUT>		
*seqNewMaster	ANTID	'segIDmaster' is the ID for the segment just defined in the Master unit.
*seqNewSlave	ANTID	'segIDslave' is the ID for the segment just defined in the Slave unit.
<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., either waveID or parmID is invalid. an2050_SENOMEM: Not enough memory. If this happens, the user can use the 'Undef' functions to free up memory occupied by definitions no longer being used.

Define Sequence `an2050_SDefineSeq`

This function is used to define a sequence.

As implied by the parameters, the segments must have been defined before the sequence can be defined. When running the sequence, the segments are run in the same order as they appear here in the parameter list.

Function Prototype:

ViStatus **`an2050_SDefineSeq`** (ViSession **vi**, ANTID ***seqID**, ANTINT16 **numSegs**, ANTID **segIDs[]**, ANTINT16 **loopMode**);

`an2050_SDefineSeq`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
numSegs	ANTINT16	Specifies the total number of segments in this sequence. Valid range: 1 to an2050_CCMAXSEGS (4096).
loopMode	ANTINT16	Specifies the loop mode of this sequence. One Shot Loop Continuously until 'stop' condition occurs.
segIDs []	ANTID	Specifies an array of 'numSegs' segment IDs that are output by functions such as 'an2050_SDefineSeg()' or 'an2050_SDefineSeg2()' and are used to identify the segments to be associated with the sequence being defined. These should be arranged in the order of running.
<OUTPUT>		
*seqID	ANTID	*seqID is the ID for the sequence just defined.
<RETURN>		
		VI_SUCCESS if successful VI_ERROR_INV_SESSION if 'vi' is invalid an2050_SEINVID: Invalid ID, i.e., 'segID' is invalid an2050_SEINVPARM: Invalid parameter. an2050_SENOMEM: Not enough memory. If this happens, the user can use the Undef functions to free up memory occupied by definitions no longer being used.

Define Sequence for DBS2055/2050A Dual Sync `an2050_SDefineSeq_2055N2050sync`

This function is used to define a sequence for use with a DBS2055.

As implied by the parameters, the segments must have been defined before the sequence can be defined. When running the sequence, the segments are run in the same order as they appear here in the parameter list.

Function Prototype:

ViStatus **`an2050_SDefineSeq_2055N2050sync`** (ViSession **viMaster**, ViSession **viSlave**, ANTID ***seqIDmaster**, ANTID ***seqIDslave**, ANTINT16 **numSegs**, ANTINT16 **loopMode**, ANTID **segIDsMaster[]**, ANTID **segIDsSlave[]**);

`An_2050_SDefineSeq_2055N2050sync`		
Parameters	**Variable Type**	**Description**
<INPUT>		
viMaster	ViSession	The session handle of the instrument to be acted upon (Master Unit). This was obtained by calling 'SetInstrumentMode'.
viSlave	ViSession	The session handle of the instrument to be acted upon (Slave unit). This was obtained by calling 'SetInstrumentMode'.
numSegs	ANTINT16	Specifies the total number of segments in this sequence. Valid range: 1 to 4096.
loopMode	ANTINT16	Specifies the loop mode of this sequence. One Shot Loop Continuously until 'stop' condition occurs.
<OUTPUT>		
*seqIDmaster	ANTID	'seqIDmaster' is the ID for the sequence just defined in the Master unit.
*seqIDslave	ANTID	'seqIDslave' is the ID for the sequence just defined in the Slave unit.
<RETURN>		
		VI_SUCCESS if successful VI_ERROR_INV_SESSION if 'vi' is invalid an2050_SEINVID: Invalid ID, i.e., 'segID' (master or slave) is invalid an2050_SEINVPARM: Invalid parameter. an2050_SENOMEM: Not enough memory. If this happens, the user can use the Undef functions to free up memory occupied by definitions no longer being used.

Define Sequence from Partial Sequence `an2050_SDefineSeqP`

This function is used to define a sequence by extracting a partial sequence from an existing sequence.

Function Prototype:

ViStatus **`an2050_SDefineSeqP`** (ViSession **vi**, ANTID **seqIDSrc**, ANTID ***seqID**, ANTINT16 **loopMode**, ANTINT16 **segBegin**, ANTINT16 **segEnd**)

`an2050_SDefineSeqP`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
seqIDSrc	ANTID	Specifies the sequence from which the partial sequence is to be extracted.
loopMode	ANTINT16	Specifies the loop mode of this sequence. One Shot Loop Continuously until 'stop' condition occurs.
segBegin	ANTID	Specifies the beginning segment to extract (i.e., start from this segment and extract all segments between this one and the one specified in segEnd). 1 = start at 1st segment, 2 = start at 2nd segment, and so on.
segEnd	ANTINT16	Specifies the ending segment. 1 for the 1st segment, 2 for the 2nd segment, and so on.
<OUTPUT>		
*seqID	ANTID	*seqID is the ID for the resulting sequence just defined.
<RETURN>		
		0 if OK.

Define Sequence from Partial Sequence DBS2055

an2050_SDefineSeqP_2055N2050sync

This function is used to define a sequence from a partial sequence for use with a DBS2055.

Function Prototype:

ViStatus **an2050_SDefineSeqP_2055N2050sync** (ViSession **viMaster**, ViSession **viSlave**, ANTID **seqIDSrcMaster**, ANTID **seqIDSrcSlave**, ANTID ***seqIDmaster**, ANTID ***seqIDslave**, ANTINT16 **loopMode**, ANTINT16 **segBeginMaster**, ANTINT16 **segBeginSlave**, ANTINT16 **segEndMaster**, ANTINT16 **segEndSlave**, ANTID **segIDsMaster[]**, ANTID **segIDsSlave[]**);

an2050_SDefineSeqP_2055N2050sync		
Parameters	**Variable Type**	**Description**
<INPUT>		
viMaster	ViSession	The session handle of the instrument to be acted upon (Master Unit). This was obtained by calling 'SetInstrumentMode'.
viSlave	ViSession	The session handle of the instrument to be acted upon (Slave unit). This was obtained by calling 'SetInstrumentMode'.
seqIDSrcMaster	ANTID	Specifies the Master unit sequence from which the partial sequence is to be extracted to form a new sequence.
seqIDSrcSlave	ANTID	Specifies the Slave unit sequence from which the partial sequence is to be extracted to form a new sequence.
loopMode	ANTINT16	Specifies the loop mode of this sequence. One Shot Loop Continuously until 'stop' condition occurs.
segBeginMaster	ANTINT16	Specifies the beginning segment of Master unit to extract, 1 for the 1st segment, 2 for the 2nd segment and so on.
segBeginSlave	ANTINT16	Specifies the beginning segment of Slave unit to extract, 1 for the 1st segment, 2 for the 2nd segment and so on.
segEndMaster	ANTINT16	Specifies the ending segment of Master unit to extract, 1 for the 1st segment, 2 for the 2nd segment and so on.
segEndSlave	ANTINT16	Specifies the ending segment of Slave unit to extract, 1 for the 1st segment, 2 for the 2nd segment and so on.
<OUTPUT>		
*seqIDmaster	ANTID	'seqIDmaster' is the ID for the Master unit resulting sequence.
*seqIDslave	ANTID	'seqIDslave' is the ID for the Slave unit resulting sequence.
<RETURN>		
		VI_SUCCESS if successful VI_ERROR_INV_SESSION if 'vi' is invalid an2050_SEINVID: Invalid ID, i.e., 'segIDSrc' (master or slave) is invalid an2050_SEINVPARM: Invalid parameter. an2050_SENOMEM: Not enough memory. If this happens, the user can use the Undef functions to free up memory occupied by definitions no longer being used.

Dual Channel

Define Dual Channel Segment `an2050_SDefineDualChnlSeg`

This function defines a segment using the data from two previously loaded waveforms: 'waveID1' and 'waveID2'. The two waveforms must have the same number of points. This function creates a new set of waveform data that contains data from 'waveID1' and waveID2' alternately (i.e., odd-numbered data points are from 'waveID1' and even-numbered data points are from 'waveID2'. The newly combined waveform data is placed in Waveform Memory. When running, 'waveID1' data is output to channel 1, and 'waveID2' data is output to channel 2. An error is returned and the function ignored if the two waves do not have the same number of data points.

Function Prototype:

ViStatus **`an2050_SDefineDualChnlSeg`** (ViSession **vi**, ANTID **waveID1**, ANTID **waveID2**, ANTID **parmID**, ANTINT16 **loopMode**, ANTID ***segID**, ANTID ***waveID**, int **mode**)

<table>
<tr><th colspan="3"><code>an2050_SDefineDualChnlSeg</code></th></tr>
<tr><th>Parameters</th><th>Variable Type</th><th>Description</th></tr>
<tr><td colspan="3"><INPUT></td></tr>
<tr><td>vi</td><td>ViSession</td><td>Specifies the instrument handle for the selected DBS2050A.</td></tr>
<tr><td>waveID1
waveID2</td><td>ANTID</td><td>Specifies the two waveforms to be combined to form the waveform data for dual channel mode. When running 'waveID1' data is output to Channel 1 and 'waveID2' data is output to Channel 2.</td></tr>
<tr><td>parmID</td><td>ANTID</td><td>Specifies the run parameters to be used.</td></tr>
<tr><td>loopMode</td><td>ANTINT16</td><td>Specifies the loop mode and loop count of the segment when running. Interpretation is as follows:
<table>
<tr><td>49152 (sets both bits below)
bit 15: Repeating Loop enable</td><td>If this bit is set, the 'loop count' (bits 0-11) is ignored. The segment repeats until 'Advance to next segment' trigger occurs.
If this bit is clear and bit 14 is clear, the segment repeats the number of times specified in 'loop count'.
If this bit is clear and bit 14 is set, the segment repeats until the advance trigger occurs or the end of loop count is reached, whichever happens first.</td></tr>
<tr><td>bit 14: Advance Trigger enable</td><td>If this bit is clear, 'advance trigger' is ignored.
If this bit is set, see above.
OR</td></tr>
<tr><td>any integer number (loop count)
bit0-bit11:</td><td>Number of loops to be repeated for the segment.
Only valid if bit 15 is clear, otherwise ignored.</td></tr>
</table>
To avoid a 'trapping' condition, make sure that if bit 15 is set, that bit 14 is not set to Ignore Advance Trigger. Otherwise the segment runs indefinitely until the entire Sequence is terminated.</td></tr>
</table>

mode	int	Specifies action if Sequencer is currently running. 0 Do not define, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and create the dual channel mode waveform, and define the segment.
<OUTPUT>		
*segID	ANTID	Outputs the segment ID for the segment just defined.
*waveID	ANTID	Outputs the waveform ID for the newly created dual channel mode waveform data.
<RETURN>		
		0 if OK.

Define Dual Channel Sequence `an2050_SDefineDualChnlSeq`

This function defines a sequence for 'dual channel mode' use. The sequence is made up of segments previously defined for 'dual channel mode' use. However, the sequence will still be defined even if segments previously defined for 'single channel mode' use are specified for use here.

A warning is returned if not all the segments specified were defined previously for dual channel mode use.

Function Prototype:

ViStatus **`an2050_SDefineDualChnlSeq`** (ViSession **vi**, ANTID ***seqID**, ANTINT16 **numSegs**, ANTINT16 **loopMode**, ANTID **segIDs[]**)

`an2050_SDefineDualChnlSeq`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
numSegs	ANTINT16	Specifies the total number of segments in this sequence. Valid range: 1 to 4096.
loopMode	ANTINT16	Specifies the loop mode of this sequence. 0 One Shot 1 Loop Continuously until 'Stop' condition occurs.
segIDs[]	ANTID	Specifies an array of 'numSegs' segment IDs that are used to identify the segments to be associated with the sequence being defined. The segments should be arranged in the order of running.
<OUTPUT>		
*seqID	ANTID	Outputs the sequence ID for the sequence just defined.
<RETURN>		
		0 if OK.

Define Dual Chnl Seg by Duplicating Wave `an2050_SDefineDualChnlSegDup`

This function duplicates the waveform data of 'waveIDin' to form the waveform data for 'dual channel mode'; the same waveform is output in both channels. The newly created dual channel waveform is then used to define a segment for dual channel mode use.

Function Prototype:

ViStatus **`an2050_SDefineDualChnlSegDup`** (ViSession **vi**, ANTID **waveIDIn**, ANTID **parmID**, ANTINT16 **loopMode**, ANTID ***segID**, ANTID ***waveIDOut**, int **mode**)

<table>
<tr><th colspan="3"><code>an2050_SDefineDualChnlSegDup</code></th></tr>
<tr><th>Parameters</th><th>Variable Type</th><th>Description</th></tr>
<tr><td colspan="3"><INPUT></td></tr>
<tr><td>vi</td><td>ViSession</td><td>Specifies the instrument handle for the selected DBS2050A.</td></tr>
<tr><td>waveIDIn</td><td>ANTID</td><td>Specifies the ID of the waveform to be duplicated to form dual channel waveform data.</td></tr>
<tr><td>parmID</td><td>ANTID</td><td>Specifies the ID of the run parameters to be used for the dual channel segment being defined.</td></tr>
<tr><td>loopMode</td><td>ANTINT16</td><td>Specifies the loop mode and loop count of the segment when running. Interpretation is as follows:
<table>
<tr><td>49152 (sets both bits below)</td><td></td></tr>
<tr><td>bit 15: Repeating Loop enable</td><td>If this bit is set, the 'loop count' (bits 0-11) is ignored. The segment repeats until 'Advance to next segment' trigger occurs.
If this bit is clear and bit 14 is clear, the segment repeats the number of times specified in 'loop count'.
If this bit is clear and bit 14 is set, the segment repeats until the advance trigger occurs or the end of loop count is reached, whichever happens first.</td></tr>
<tr><td>bit 14: Advance Trigger enable</td><td>If this bit is clear, 'advance trigger' is ignored.
If this bit is set, see above.
OR</td></tr>
<tr><td>any integer number (loop count)
bit0-bit11:</td><td>Number of loops to be repeated for the segment.
Only valid if bit 15 is clear, otherwise ignored.</td></tr>
</table>
To avoid a 'trapping' condition, make sure that if bit 15 is set, that bit 14 is not set to Ignore Advance Trigger. Otherwise the segment will run indefinitely until the entire Sequence is terminated.</td></tr>
<tr><td>mode</td><td>int</td><td>Specifies action if Sequencer is currently running.
0 Do not define, return an2050_SESQNCRRUNNING
1 Stop the Sequencer and create the dual channel mode waveform, and define the segment.</td></tr>
<tr><td colspan="3"><OUTPUT></td></tr>
<tr><td>*segID</td><td>ANTID</td><td>Specifies the segment ID for the segment just defined.</td></tr>
<tr><td>*waveIDOut</td><td>ANTID</td><td>Specifies the waveform ID for the newly created dual channel mode waveform data.</td></tr>
</table>

<RETURN>		
		0 if OK.

Define Dual Chnl Seg. from Seg `an2050_SDefineDualChnlSegFromSeg`

This function duplicates the waveform data of 'segIDIn' to create a new dual channel mode waveform and segment defined for dual channel mode use. The 'parms' and 'loopMode' of 'segIDIn' are also used for the new segment, 'segIDOut', for dual channel mode.

Function Prototype:

ViStatus **`an2050_SDefineDualChnlSegFromSeg`** (ViSession **vi**, ANTID **segIDIn**, ANTID ***segIDOut**, ANTID ***waveID**, int **mode**)

`an2050_SDefineDualChnlSegFromSeg`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
segIDIn	ANTID	Specifies the ID of the segment which includes the waveform to be duplicated to form dual channel mode waveform data. The run parameters and loop mode of this segment is used for the new segment being created.
mode	int	Specifies action if Sequencer is currently running. 0 Do not define, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and create the dual channel mode waveform and define the segment.
<OUTPUT>		
*segIDOut	ANTID	Specifies the segment ID for the segment just defined.
*waveID	ANTID	Specifies the waveform ID for the newly created dual channel mode waveform data.
<RETURN>		
		0 if OK.

Define Dual Channel Segment1 `an2050_SDefineDualChnlSeg1`

This function defines a segment for dual channel use. Normally, the 'waveID' is one specifying a waveform previously defined for dual channel mode use, possibly from one of the following functions:

- 'an2050_LoadDualChnlWave()',
- 'an2050_LoadDualChnlWaves()',
- 'an2050_LoadDualChnlWaveDup()', or
- 'an2050_SDefineDualChnlSeg*x*()'

However, it is possible to specify a waveID previously defined for single channel mode use. The function will issue a warning indicating that the waveform data is not for 'dual channel mode'.

Function Prototype:

ViStatus **`an2050_SDefineDualChnlSeg1`** (ViSession **vi**, ANTID **waveID**, ANTID **parmID**, ANTINT16 **loopMode**, ANTID ***segID**)

<table>
<tr><th colspan="3"><code>an2050_SDefineDualChnlSeg1</code></th></tr>
<tr><th>Parameters</th><th>Variable Type</th><th>Description</th></tr>
<tr><td colspan="3"><INPUT></td></tr>
<tr><td>vi</td><td>ViSession</td><td>Specifies the instrument handle for the selected DBS2050A.</td></tr>
<tr><td>waveID</td><td>ANTID</td><td>Specifies the ID of the dual channel mode waveform.</td></tr>
<tr><td>parmID</td><td>ANTID</td><td>Specifies the ID of the run parameters to be used for the dual channel segment being defined.</td></tr>
<tr><td>loopMode</td><td>ANTINT16</td><td>Specifies the loop mode and loop count of the segment when running. Interpretation is as follows:
<table>
<tr><td>49152 (sets both bits below)</td><td></td></tr>
<tr><td>bit 15: Repeating Loop enable</td><td>If this bit is set, the 'loop count' (bits 0-11) is ignored. The segment repeats until 'Advance to next segment' trigger occurs.
If this bit is clear and bit 14 is clear, the segment repeats the number of times specified in 'loop count'.
If this bit is clear and bit 14 is set, the segment repeats until the advance trigger occurs or the end of loop count is reached, whichever happens first.</td></tr>
<tr><td>bit 14: Advance Trigger enable</td><td>If this bit is clear, 'advance trigger' is ignored.
If this bit is set, see above.
OR</td></tr>
<tr><td>any integer number (loop count)
bit0-bit11:</td><td>Number of loops to be repeated for the segment.
Only valid if bit 15 is clear, otherwise ignored.</td></tr>
</table>
To avoid a 'trapping' condition, make sure that if bit 15 is set, that bit 14 is not set to Ignore Advance Trigger. Otherwise the segment will run indefinitely until the entire Sequence is terminated.</td></tr>
</table>

<OUTPUT>		
*segID	ANTID	Specifies the segment ID for the segment just defined.
<RETURN>		
		0 if OK.

DBS2055 and DBS2050A Dual Synchronous Segment and Sequence Definition

Set and Start Sequence `an2050_SSetSeqM_2055N2050sync`

This function loads the specified sequence into the Sequencer Memory (if not already loaded), sets the desired markers as specified by 'm1Master', 'm2Master', 'm1ArrayMaster' and 'm2ArrayMaster' and similar Slave parameters and enables the running of the sequence.

In normal marker mode, when a segment has marker(s) turned on, the marker pulse will appear at 32 clock periods times the associated marker position (set through the function 'an2050_GConfigMarkers) into the segment, and if the segment is looped in any way, the marker pulse will appear in every loop.

An error will be returned and nothing done if there is not enough memory in Sequencer Memory to load the sequence.

Function Prototype:

ViStatus **`an2050_SSetSeqM_2055N2050sync`** (ViSession **viMaster**, ViSession **viSlave**, ANTID **seqIDmaster**, ANTID **seqIDslave**, int **m1Master**, int **m2Master**, char **m1ArrayMaster[]**, char **m2ArrayMaster[]**, int **m1Slave**, int **m2Slave**, char **m1ArraySlave[]**, char **m2ArraySlave[]**, ANTID ***seqNewMaster**, ANTID ***seqNewSlave**, int **modeMaster**, int **modeSlave**)

`an2050_SSetSeqM_2055N2050sync`		
Parameters	**Variable Type**	**Description**
<INPUT>		
viMaster	ViSession	The session handle of the instrument to be acted upon (Master unit). This was obtained by calling 'SetInstrumentMode'.
viSlave	ViSession	The session handle of the instrument to be acted upon (Slave unit). This was obtained by calling 'SetInstrumentMode'.
seqIDmaster	ANTID	The sequence ID of the sequence to be run. An invalid ID will cause an error return and no action.
seqIDslave	ANTID	The sequence ID of the sequence to be run. An invalid ID will cause an error return and no action.
m1Master	int	Marker enable for markers on the Master unit. 0 Turn Marker 1 OFF for all segments. 1 Turn Marker 1 ON for all segments. 2 Turn Marker 1 ON or OFF according to m1ArrayMaster[] All others cause 'AN2050_SEINVPARM' to return.
m2Master	int	Marker enable for markers on the Master unit. 0 Turn Marker 2 OFF for all segments. 1 Turn Marker 2 ON for all segments. 2 Turn Marker 2 ON or OFF according to m2ArrayMaster[] All others cause 'AN2050_SEINVPARM' to return.
m1ArrayMaster[]	char	This parameter is used only if m1Master is 2. When used, each element specifies whether Marker 1 of the corresponding segment (i.e., 'm1ArrayMaster[0]' specifies the first segment, 'm1ArrayMaster[1]' specifies the second segment, and so on) should be turned ON (1), or OFF (0). Values other than 0 or 1 will cause the return of 'AN2050_GEINVPARM'.

		The size of this array must be at least the total number of segments in the sequence 'seqIDMaster' (this may be obtained by calling 'an2050_SGetSeqInfo'). This parameter is ignored if m1Master is not 2.
m2ArrayMaster[]	char	This parameter is used only if m2Master is 2. When used, each element specifies whether Marker 2 of the corresponding segment (i.e., 'm2ArrayMaster[0]' specifies the first segment, 'm2ArrayMaster[1]' specifies the second segment, and so on) should be turned ON (1), or OFF (0). Values other than 0 or 1 will cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqIDMaster' (this may be obtained by calling 'an2050_SGetSeqInfo'). This parameter is ignored if m2Master is not 2.
m1Slave	int	Marker enable for markers on the Slave unit. 0 Turn Marker 1 OFF for all segments. 1 Turn Marker 1 ON for all segments. 2 Turn Marker 1 ON or OFF according to m1ArraySlave[] All others cause 'AN2050_SEINVPARM' to return.
m2Slave	int	Marker enable for markers on the Slave unit. 0 Turn Marker 2 OFF for all segments. 1 Turn Marker 2 ON for all segments. 2 Turn Marker 2 ON or OFF according to m2ArraySlave[] All others cause 'AN2050_SEINVPARM' to return.
m1ArraySlave[]	char	This parameter is used only if m1Slave is 2. When used, each element specifies whether Marker 1 of the corresponding segment (i.e., 'm1ArraySlave[0]' specifies the first segment, 'm1ArraySlave[1]' specifies the second segment, and so on) should be turned ON (1), or OFF (0). Values other than 0 or 1 will cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqIDSlave' (this may be obtained by calling 'an2050_SGetSeqInfo'). This parameter is ignored if m1Slave is not 2.
m2ArraySlave[]	char	This parameter is used only if m2Slave is 2. When used, each element specifies whether Marker 2 of the corresponding segment (i.e., 'm2ArraySlave[0]' specifies the first segment, 'm2ArraySlave[1]' specifies the second segment, and so on) should be turned ON (1), or OFF (0). Values other than 0 or 1 will cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqIDSlave' (this may be obtained by calling 'an2050_SGetSeqInfo'). This parameter is ignored if m2Slave is not 2.
modeMaster	int	This parameter dictates function behavior if the sequencer is already running. Bits 0 to 3: Specifies action if 'Run Sequencer' is currently enabled. Do not set, return with 'AWSESQNCRRUNNING' to indicate that the sequencer is running. Stop the sequencer and set the sequence. Bits 4 to 7: Specifies action if the sequence is loaded already. Use the loaded sequence, and ignore the marker definition specified above. Use the loaded sequence, but use the marker definition as specified above. Keep the loaded sequence, make a copy of it and load the new sequence with the new marker definition. If this is specified and 'seqIDmaster' is already loaded, the ID of the new sequence copy will be output in 'seqNewMaster'.
modeSlave	int	This parameter dictates function behavior if the sequencer is already running. Bits 0 to 3: Specifies action if 'Run Sequencer' is currently enabled. Do not set, return with 'AWSESQNCRRUNNING' to indicate that the sequencer is running. Stop the sequencer and set the sequence. Bits 4 to 7: Specifies action if the sequence is loaded already.

		Use the loaded sequence, and ignore the marker definition specified above. Use the loaded sequence, but use the marker definition as specified above. Keep the loaded sequence, make a copy of it and load the new sequence with the new marker definition. If this is specified and 'seqIDslave' is already loaded, the ID of the new sequence copy will be output in 'seqNewSlave'.
<OUTPUT>		
*seqNewMaster	ANTID	This parameter is the Master Unit ID of the new copy of 'seqIDmaster' if 'seqIDmaster' is already loaded, and ('mode'&0xf0==0x20). '*seqNewMaster' will be 0 if no new copy of 'seqIDmaster' is created (i.e., 'seqIDmaster' was not loaded when this function was called). '*seqNewMaster' is meaningful only if 'mode'*0xf0==0x20, otherwise it contains only garbage.
*seqNewSlave	ANTID	This parameter is the Slave Unit ID of the new copy of 'seqIDslave' if 'seqIDslave' is already loaded, and ('mode'&0xf0==0x20). '*seqNewMaster' will be 0 if no new copy of 'seqIDslave' is created (i.e., 'seqIDslave' was not loaded when this function was called). '*seqNewMaster' is meaningful only if 'mode'*0xf0==0x20, otherwise it contains only garbage.
<RETURN>		
		VI_SUCCESS if successful. an2050_SEINVID: Invalid ID used in function call. VI_ERROR_INV_SESSION: if 'viMaster' or 'viSlave' is invalid.

Set and Start Sequence `an2050_SSetSeqMN_2055N2050sync`

This function loads the specified sequence into the Sequencer Memory (if not already loaded), sets the desired markers as specified by 'm1Master', 'm2Master', 'm3Master', 'm1ArrayMaster' and 'm2ArrayMaster' and similar Slave parameters.

In normal marker mode, when a segment has marker(s) turned on, the marker pulse will appear at 32 clock periods times the associated marker position (set through the function 'an2050_GConfigMarkers) into the segment, and if the segment is looped in any way, the marker pulse will appear in every loop.

An error will be returned and nothing done if there is not enough memory in Sequencer Memory to load the sequence.

Function Prototype:

ViStatus **`an2050_SsetSeqMN_2055N2050sync`** (ViSession **viMaster**, ViSession **viSlave**, ANTID **seqIDmaster**, ANTID **seqIDslave**, int **m1Master**, int **m2Master**, char **m1ArrayMaster[]**, char **m2ArrayMaster[]**, int **m1Slave**, int **m2Slave**, char **m1ArraySlave[]**, char **m2ArraySlave[]**, ANTID ***seqNewMaster**, ANTID ***seqNewSlave**, int **modeMaster**, int **modeSlave**)

`an2050_SSetSeqMN_2055N2050sync`		
Parameters	**Variable Type**	**Description**
<INPUT>		
viMaster	ViSession	The session handle of the instrument to be acted upon (Master unit). This was obtained by calling 'SetInstrumentMode'.
viSlave	ViSession	The session handle of the instrument to be acted upon (Slave unit). This was obtained by calling 'SetInstrumentMode'.
seqIDmaster	ANTID	The sequence ID of the sequence to be run. An invalid ID will cause an error return and no action.
seqIDslave	ANTID	The sequence ID of the sequence to be run. An invalid ID will cause an error return and no action.
m1Master	int	Marker enable for markers on the Master unit. 0 Turn Marker 1 OFF for all segments. 1 Turn Marker 1 ON for all segments. 2 Turn Marker 1 ON or OFF according to m1ArrayMaster[] All others cause 'AN2050_SEINVPARM' to return.
m2Master	int	Marker enable for markers on the Master unit. 0 Turn Marker 2 OFF for all segments. 1 Turn Marker 2 ON for all segments. 2 Turn Marker 2 ON or OFF according to m2ArrayMaster[] All others cause 'AN2050_SEINVPARM' to return.
m3Master	int	Marker enable for markers on the Master unit. 0 Turn Marker 2 OFF for all segments. 1 Turn Marker 2 ON for all segments. 2 Turn Marker 2 ON or OFF according to m3ArrayMaster[] All others cause 'AN2050_SEINVPARM' to return.

m1ArrayMaster[]	char	This parameter is used only if m1Master is 2. When used, each element specifies whether Marker 1 of the corresponding segment (i.e., 'm1ArrayMaster[0]' specifies the first segment, 'm1ArrayMaster[1]' specifies the second segment, and so on) should be turned ON (1), or OFF (0). Values other than 0 or 1 will cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqIDMaster' (this may be obtained by calling 'an2050_SGetSeqInfo'). This parameter is ignored if m1Master is not 2.
m2ArrayMaster[]	char	This parameter is used only if m2Master is 2. When used, each element specifies whether Marker 2 of the corresponding segment (i.e., 'm2ArrayMaster[0]' specifies the first segment, 'm2ArrayMaster[1]' specifies the second segment, and so on) should be turned ON (1), or OFF (0). Values other than 0 or 1 will cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqIDMaster' (this may be obtained by calling 'an2050_SGetSeqInfo'). This parameter is ignored if m2Master is not 2.
m3ArrayMaster[]	char	This parameter is used only if m3Master is 2. When used, each element specifies whether Marker 3 of the corresponding segment (i.e., 'm3ArrayMaster[0]' specifies the first segment, 'm3ArrayMaster[1]' specifies the second segment, and so on) should be turned ON (1), or OFF (0). Values other than 0 or 1 will cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqIDMaster' (this may be obtained by calling 'an2050_SGetSeqInfo'). This parameter is ignored if m3Master is not 2.
m1Slave	int	Marker enable for markers on the Slave unit. 0 Turn Marker 1 OFF for all segments. 1 Turn Marker 1 ON for all segments. 2 Turn Marker 1 ON or OFF according to m1ArraySlave[] All others cause 'AN2050_SEINVPARM' to return.
m2Slave	int	Marker enable for markers on the Slave unit. 0 Turn Marker 2 OFF for all segments. 1 Turn Marker 2 ON for all segments. 2 Turn Marker 2 ON or OFF according to m2ArraySlave[] All others cause 'AN2050_SEINVPARM' to return.
m3Slave	int	Marker enable for markers on the Slave unit. 0 Turn Marker 2 OFF for all segments. 1 Turn Marker 2 ON for all segments. 2 Turn Marker 2 ON or OFF according to m3ArraySlave[] All others cause 'AN2050_SEINVPARM' to return.
m1ArraySlave[]	int	This parameter is used only if m1Slave is 2. When used, each element specifies whether Marker 1 of the corresponding segment (i.e., 'm1ArraySlave[0]' specifies the first segment, 'm1ArraySlave[1]' specifies the second segment, and so on) should be turned ON (1), or OFF (0). Values other than 0 or 1 will cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqIDSlave' (this may be obtained by calling 'an2050_SGetSeqInfo'). This parameter is ignored if m1Slave is not 2.
m2ArraySlave[]	int	This parameter is used only if m2Slave is 2. When used, each element specifies whether Marker 2 of the corresponding segment (i.e., 'm2ArraySlave[0]' specifies the first segment, 'm2ArraySlave[1]' specifies the second segment, and so on) should be turned ON (1), or OFF (0). Values other than 0 or 1 will cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqIDSlave' (this may be obtained by calling 'an2050_SGetSeqInfo'). This parameter is ignored if m2Slave is not 2.

m3ArraySlave[]	int	This parameter is used only if m3Slave is 2. When used, each element specifies whether Marker 3 of the corresponding segment (i.e., 'm3ArraySlave[0]' specifies the first segment, 'm3ArraySlave[1]' specifies the second segment, and so on) should be turned ON (1), or OFF (0). Values other than 0 or 1 will cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqIDSlave' (this may be obtained by calling 'an2050_SGetSeqInfo'). This parameter is ignored if m3Slave is not 2.
modeMaster	int	This parameter dictates function behavior if the sequencer is already running. Bits 0 to 3: Specifies action if 'Run Sequencer' is currently enabled. Do not set, return with 'AWSESQNCRRUNNING' to indicate that the sequencer is running. Stop the sequencer and set the sequence. Bits 4 to 7: Specifies action if the sequence is loaded already. Use the loaded sequence, and ignore the marker definition specified above. Use the loaded sequence, but use the marker definition as specified above. Keep the loaded sequence, make a copy of it and load the new sequence with the new marker definition. If this is specified and 'seqIDmaster' is already loaded, the ID of the new sequence copy will be output in 'seqNewMaster'.
modeSlave	int	This parameter dictates function behavior if the sequencer is already running. Bits 0 to 3: Specifies action if 'Run Sequencer' is currently enabled. Do not set, return with 'AWSESQNCRRUNNING' to indicate that the sequencer is running. Stop the sequencer and set the sequence. Bits 4 to 7: Specifies action if the sequence is loaded already. Use the loaded sequence, and ignore the marker definition specified above. Use the loaded sequence, but use the marker definition as specified above. Keep the loaded sequence, make a copy of it and load the new sequence with the new marker definition. If this is specified and 'seqIDslave' is already loaded, the ID of the new sequence copy will be output in 'seqNewSlave'.
<OUTPUT>		
*seqNewMaster	ANTID	This parameter is the Master Unit ID of the new copy of 'seqIDmaster' if 'seqIDmaster' is already loaded, and ('mode'&0xf0==0x20). '*seqNewMaster' will be 0 if no new copy of 'seqIDmaster' is created (i.e., 'seqIDmaster' was not loaded when this function was called). '*seqNewMaster' is meaningful only if 'mode'*0xf0==0x20, otherwise it contains only garbage.
*seqNewSlave	ANTID	This parameter is the Slave Unit ID of the new copy of 'seqIDslave' if 'seqIDslave' is already loaded, and ('mode'&0xf0==0x20). '*seqNewMaster' will be 0 if no new copy of 'seqIDslave' is created (i.e., 'seqIDslave' was not loaded when this function was called). '*seqNewMaster' is meaningful only if 'mode'*0xf0==0x20, otherwise it contains only garbage.
<RETURN>		
		VI_SUCCESS if successful. an2050_SEINVID: Invalid ID used in function call. VI_ERROR_INV_SESSION: if 'viMaster' or 'viSlave' is invalid.

Run-Time Parameters

Define Run-Time Parameters `an2050_SDefineRunParms`

This function registers a set of Run-Time Parameters for later use (mainly in defining segment and/or sequence).

If an 'Out of memory' error occurs, use 'Undefine' commands to free some memory used by obsolete 'parameter', 'segment' or 'sequence' definitions.

Function Prototype:

ViStatus **`an2050_SDefineRunParms`** (ViSession **vi**, ANTFLOAT **atten1**, ANTFLOAT **atten2**, ANTFLOAT **offset1**, ANTFLOAT **offset2**, ANTID ***parmID**)

`an2050_SDefineRunParms`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
atten1 atten2	ANTFLOAT	Specifies the attenuation desired for Channel 1 and Channel 2. Valid range 0 to –63dB. The attenuator has a 12-bit resolution. The software rounds off to the nearest achievable attenuation.
offset1 offset 2	ANTFLOAT	Specifies the offset for Channel 1 and Channel 2. Valid range –3.5V to +3.5V. 12-bit resolution.
<OUTPUT>		
*parmID	ANTID	ParmID is a unique number to represent the Run Parameters just defined.
<RETURN>		
		0 if OK. an2050_SENOMEM: Not enough memory. If this happens, the user can use the undefine functions (e.g., 'an2050_SUndefSeg()') to free memory occupied by definitions no longer used.

Define Run-Time Parameters for DBS2055 `an2050_SDefineRunParms_2055`

This function registers a set of Run-Time Parameters for later use (mainly in defining segment and/or sequence) with a DBS2055.

If an 'Out of memory' error occurs, use 'Undefine' commands to free some memory used by obsolete 'parameter', 'segment' or 'sequence' definitions.

Function Prototype:

ViStatus **`an2050_SDefineRunParms_2055`** (ViSession **viMaster**, ViSession **viSlave**, ANTFLOAT **atten1**, ANTFLOAT **atten2**, ANTFLOAT **offset1**, ANTFLOAT **offset2**, ANTID ***parmIDmaster**, ANTID ***parmIDslave**)

`an2050_SDefineRunParms_2055`		
Parameters	**Variable Type**	**Description**
<INPUT>		
viMaster	ViSession	The session handle of the instrument to be acted upon (Master unit). This was obtained by calling 'SetInstrumentMode'.
viSlave	ViSession	The session handle of the instrument to be acted upon (Slave unit). This was obtained by calling 'SetInstrumentMode'.
atten1	ANTFLOAT	The attenuation desired for channel 2. Valid range 0 to –63 dB. The attenuator has 12-bit resolution. The software will round off to the nearest achievable attenuation.
atten2	ANTFLOAT	The attenuation desired for channel 1. Valid range 0 to –63 dB. The attenuator has 12-bit resolution. The software will round off to the nearest achievable attenuation.
offset1	ANTFLOAT	The offset for channel 1 in volts. Valid range: -3.5V to +3.5V.
offset2	ANTFLOAT	The offset for channel 2 in volts. Valid range: -3.5V to +3.5V.
<OUTPUT>		
*parmIDmaster	ANTID	'*parmIDmaster' is the Master unit ID for the Run-Time Parameters just defined.
*parmIDslave	ANTID	'*parmIDslave' is the Slave unit ID for the Run-Time Parameters just defined.
<RETURN>		
		0 if OK. an2050_SENOMEM: Not enough memory. If this happens, the user can use the undefine functions (e.g., 'an2050_SUndefSeg()') to free memory occupied by definitions no longer used.

Unload and Undefine

These functions unload and undefine a specified waveform, sequence, run time parameter, or segment. The following functions are included:

- an2050_SUnloadWaveform
- an2050_SUnloadWaveform_2055N2050sync
- an2050_SUnloadSeq
- an2050_SUnloadSeq_2055N2050sync
- an2050_SUnloadAllSeqs
- an2050_SUndefRunParms
- an2050_SUndefRunParms_2055N2050sync
- an2050_SUndefSeg
- an2050_SUndefSeg_2055N2050sync
- an2050_SUndefSeq
- an2050_SUndefSeq_2055N2050sync
- an2050_SUndefAllDefs

Unload

Unload Waveform

an2050_SUnloadWaveform

This function unloads the specified waveform from Waveform Memory by calling 'an2050_MUnloadWaveform()' and also performs housekeeping work for the Sequencer Manager, i.e., undefines the segments and sequences that use this waveform and unloads the currently loaded sequences that use this waveform from Sequence Memory.

Function Prototype:

ViStatus **an2050_SUnloadWaveform** (ViSession **vi**, ANTID **waveID**, int **mode**)

an2050_SUnloadWaveform		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
waveID	ANTID	Specifies the ID of the waveform to be unloaded. An invalid ID cause error return and nothing is done. If ALL waveforms are to be unloaded use 0.
mode	int	Specifies action if Sequencer is currently running. 0 Do not unload, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and unload the waveform.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK.

Unload Waveform for DBS2055 `an2050_SUnloadWaveform_2055N2050sync`

This function unloads the specified waveform from waveform data memory and also performs housekeeping work for the Sequence Manager – unloads the sequences that use this waveform and are currently loaded from Sequencer Memory.

Function Prototype:

ViStatus **`an2050_SUnloadWaveform_2055N2050sync`** (ViSession **viMaster**, ViSession **viSlave**, ANTID **waveIDmaster**, ANTID **waveIDslave**, int **mode**)

`an2050_SUnloadWaveform_2055N2050sync`		
Parameters	**Variable Type**	**Description**
<INPUT>		
viMaster	ViSession	The session handle of the instrument to be acted upon (Master unit). This was obtained by calling 'SetInstrumentMode'.
viSlave	ViSession	The session handle of the instrument to be acted upon (Slave unit). This was obtained by calling 'SetInstrumentMode'.
waveIDmaster	ANTID	The Master unit ID of the waveform to be unloaded. An invalid ID will cause this command to be ignored.
waveIDslave	ANTID	The Slave unit ID of the waveform to be unloaded. An invalid ID will cause this command to be ignored.
mode	int	Specifies action if Sequencer is currently running. 0 Do not undefine, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and undefine the run parameters.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., 'segID' is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., 'run sequencer' is enabled).

Unload Sequence `an2050_SUnloadSeq`

This function unloads a sequence 'seqID' from Sequencer Memory and frees the memory previously occupied by this sequence.

CAUTION
A sequence may be used both by regular sequence runs and by branch vectors. Be sure a sequence is no longer needed by either regular sequence run or branch vectors before unloading it.

Function Prototype:

ViStatus **`an2050_SUnloadSeq`** (ViSession **vi**, ANTID **seqID**, int **mode**)

`an2050_SUnloadSeq`		
Parameters		**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
seqID	ANTID	Specifies the ID of the sequence to be unloaded. An invalid ID will cause this command to be ignored.
mode	int	Specifies action if Sequencer is currently running. 0 Do not unload, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and unload the sequence.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., 'seqID' is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., 'run sequencer' is enabled).

Unload Sequence for DBS2055 `an2050_SUnLoadSeq_2055N2050sync`

This function unloads the sequences ‘seqIDmaster’ and ‘seqIDslave’ from Sequencer Memory, and frees the memory previously occupied by the segments of these sequences.

Function Prototype:

ViStatus **`an2050_SUnloadSeq_2055N2050sync`** (ViSession **viMaster**, ViSession **viSlave**, ANTID **seqIDmaster**, ANTID **seqIDslave**, int **mode**)

`an2050_SUnloadSeq_2055N2050sync`		
Parameters	**Variable Type**	**Description**
<INPUT>		
viMaster	ViSession	The session handle of the instrument to be acted upon (Master unit). This was obtained by calling ‘SetInstrumentMode’.
viSlave	ViSession	The session handle of the instrument to be acted upon (Slave unit). This was obtained by calling ‘SetInstrumentMode’.
seqIDmaster	ANTID	The Master unit ID of the sequence to be unloaded. An invalid ID will cause this command to be ignored.
seqIDslave	ANTID	The Slave unit ID of the sequence to be unloaded. An invalid ID will cause this command to be ignored.
mode	int	Specifies action if Sequencer is currently running. 0 Do not undefine, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and undefine the run parameters.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., ‘segID’ is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., ‘run sequencer’ is enabled).

Unload All Sequences — `an2050_SUnloadAllSeqs`

This function unloads ALL the sequences currently in Sequencer Memory. This makes the entire Sequencer Memory available for loading sequences.

This function and all 'an2050_SUndefAll*XXX*()' functions render all branch vectors inappropriate or invalid.

Function Prototype:

ViStatus **`an2050_SUnloadAllSeqs`** (ViSession **vi**, int **mode**)

`an2050_SUnloadAllSeqs`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
mode	int	Specifies action if Sequencer is currently running. 0 Do not unload, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and unload all sequences.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., 'seqID' is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., 'run sequencer' is enabled).

Undefine

Undefine Run Parameters `an2050_SUndefRunParms`

This function is used to remove the 'run parameters' identified by 'parmID' from memory.

This action also causes all segments and sequences that reference this 'parmID' to be undefined, and if they are currently loaded in the Sequencer Memory, they are unloaded.

Since Sequencer Memory is not accessible when running, this command will not be executed if the Sequencer is running, unless 'mode' indicates otherwise (see below).

Function Prototype:

ViStatus **`an2050_SUndefRunParms`** (ViSession **vi**, ANTID **parmID**, int **mode**)

`an2050_SUndefRunParms`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
parmID	ANTID	Specifies the ID of the run parameters to be undefined. An invalid ID will cause this command to be ignored.
mode	int	Specifies action if Sequencer is currently running. 0 Do not undefine, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and undefine the run parameters.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., 'parmID' is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., 'run sequencer' is enabled).

Undefine Run Parameters for DBS2055 `an2050_SUndefRunParms_2055N2050sync`

This function is used to remove the 'run parameters' identified by 'parmID' (master and slave) from memory.

This action causes all segments and sequences that reference this 'parmID' to also be undefined, and if they are currently loaded in the Sequencer Memory, they are unloaded.

Since Sequencer Memory is not accessible when running, this command will not be executed if the Sequencer is running, unless 'mode' indicates otherwise (see below).

Function Prototype:

ViStatus **`an2050_SundefRunParms_2055N2050sync`** (ViSession **viMaster**, ViSession **viSlave**, ANTID **parmIDmaster**, ANTID **parmIDslave**, int **mode**)

`an2050_SundefRunParms_2055N2050sync`		
Parameters	**Variable Type**	**Description**
<INPUT>		
viMaster	ViSession	The session handle of the instrument to be acted upon (Master unit). This was obtained by calling 'SetInstrumentMode'.
viSlave	ViSession	The session handle of the instrument to be acted upon (Slave unit). This was obtained by calling 'SetInstrumentMode'.
parmIDmaster	ANTID	The Master unit ID of the run parameters to be undefined. An invalid ID will cause this command to be ignored.
parmIDslave	ANTID	The Slave unit ID of the run parameters to be undefined. An invalid ID will cause this command to be ignored.
mode	Int	Specifies action if Sequencer is currently running. 0 Do not undefine, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and undefine the run parameters.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., 'parmID' is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., 'run sequencer' is enabled).

Undefine Segment `an2050_SUndefSeg`

This function is used to remove the segment identified by 'segID' from Sequencer memory. This will also cause the sequences that reference this 'segID' to be undefined, and if they are currently loaded in Sequencer Memory, they are unloaded.

Since Sequencer Memory is not accessible when running, this command will not be executed if the Sequencer is running unless 'mode' indicates otherwise (see below).

Function Prototype:

ViStatus **`an2050_SUndefSeg`** (ViSession **vi**, ANTID **segID**, int **mode**)

`an2050_SUndefSeg`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
segID	ANTID	Specifies the ID of the segment to be undefined. An invalid ID will cause this command to be ignored.
mode	int	Specifies action if Sequencer is currently running. 0 Do not undefine, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and undefine the segment.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., 'segID' is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., 'run sequencer' is enabled).

Undefine Segment for DBS2055 `an2050_SUndefSeg_2055N2050sync`

This function is used to remove the segment identified by 'segID' (master and slave) from memory.

This action causes all sequences that reference this 'segID' to also be undefined, and if they are currently loaded in the Sequencer Memory, they are unloaded.

Since Sequencer Memory is not accessible when running, this command will not be executed if the Sequencer is running, unless 'mode' indicates otherwise (see below).

Although leaving behind too many useless segments will unnecessarily decrease the performance of the instrument, normally it is not necessary to undefined a segment.

Function Prototype:

ViStatus **`an2050_SUndefSeg_2055N2050sync`** (ViSession **viMaster**, ViSession **viSlave**, ANTID **segIDmaster**, ANTID **segIDslave**, int **mode**)

`an2050_SUndefSeg_2055N2050sync`		
Parameters	**Variable Type**	**Description**
<INPUT>		
viMaster	ViSession	The session handle of the instrument to be acted upon (Master unit). This was obtained by calling 'SetInstrumentMode'.
viSlave	ViSession	The session handle of the instrument to be acted upon (Slave unit). This was obtained by calling 'SetInstrumentMode'.
segIDmaster	ANTID	The Master unit ID of the segment to be undefined. An invalid ID will cause this command to be ignored.
segIDslave	ANTID	The Slave unit ID of the segment to be undefined. An invalid ID will cause this command to be ignored.
mode	int	Specifies action if Sequencer is currently running. 0 Do not undefine, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and undefine the run parameters.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., 'segID' is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., 'run sequencer' is enabled).

Undefine Sequence `an2050_SUndefSeq`

This function undefines the sequence identified by 'seqID'. This will free the memory occupied by the sequence. If the sequence is currently loaded in Sequence Memory, it is unloaded.

Since Sequencer Memory is not accessible when running, this command will not be executed if the Sequencer is running unless 'mode' indicates otherwise (see below).

Function Prototype:

ViStatus **`an2050_SUndefSeq`** (ViSession **vi**, ANTID **seqID**, int **mode**)

`an2050_SUndefSeq`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
seqID	ANTID	Specifies the ID of the sequence to be undefined. An invalid ID cause this command to be ignored.
mode	int	Specifies action if Sequencer is currently running. 0 Do not undefine, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and undefine the sequence.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., 'seqID' is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., 'run sequencer' is enabled).

Undefine Sequence for DBS2055 `an2050_SUndefSeq_2055N2050sync`

This function is used to remove the segment identified by 'seqID' (master and slave) from memory.

This action also causes all sequences that reference this 'seqID' to be undefined, and if they are currently loaded in the Sequencer Memory, they are unloaded.

Since Sequencer Memory is not accessible when running, this command will not be executed if the Sequencer is running, unless 'mode' indicates otherwise (see below).

Although leaving behind too many useless sequences will unnecessarily decrease the performance of the instrument, normally it is not necessary to undefined a sequence.

Function Prototype:

ViStatus **`an2050_SUndefSeq_2055N2050sync`** (ViSession **viMaster**, ViSession **viSlave**, ANTID **seqIDmaster**, ANTID **seqIDslave**, int **mode**)

`an2050_SUndefSeq_2055N2050sync`		
Parameters	**Variable Type**	**Description**
<INPUT>		
viMaster	ViSession	The session handle of the instrument to be acted upon (Master unit). This was obtained by calling 'SetInstrumentMode'.
viSlave	ViSession	The session handle of the instrument to be acted upon (Slave unit). This was obtained by calling 'SetInstrumentMode'.
seqIDmaster	ANTID	The Master unit ID of the sequence to be undefined. An invalid ID will cause this command to be ignored.
seqIDslave	ANTID	The Slave unit ID of the sequence to be undefined. An invalid ID will cause this command to be ignored.
mode	int	Specifies action if Sequencer is currently running. 0 Do not undefine, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and undefine the run parameters.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., 'segID' is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., 'run sequencer' is enabled).

Undefine All Definitions `an2050_SUndefAllDefs`

This function clears all definitions. This function renders branch vectors invalid.

Function Prototype:

ViStatus **`an2050_SUndefAllDefs`** (ViSession **vi**, int **level**, int **mode**)

`an2050_SUndefAllDefs`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
level	int	Specifies definitions to be undefined. Valid range is 1-3, invalid values cause error return and nothing done. 1 Specifies that ALL defined sequences are to be undefined. 2 Specifies that ALL defined segments are to be undefined. This, by definition, also undefines ALL sequences. 3 Specifies that ALL defined run parameters are to be undefined. This, by definition, also undefines ALL segments and sequences.
mode	int	Specifies action if Sequencer is currently running. 0 Do not undefine, return an2050_SESQNCRRUNNING 1 Stop the Sequencer and undefine all requested.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_SEINVID: Invalid ID, i.e., 'seqID' is invalid. an2050_SEINVPARM: Invalid ID, i.e., 'parmID' is invalid. an2050_SESQNCRRUNNING: Sequencer is running (i.e., 'run sequencer' is enabled).

Trigger

These functions control the main, advance and branch triggers. The following functions are included:

- an2050_CSetRunTrigMode
- an2050_CSetTrigLevel
- an2050_GSetAdvanceTrigMode
- an2050_GTrigger
- an2050_SSetBranchVectorMN
- an2050_SSetBranchVector
- an2050_SSetBranchVectorM
- an2050_GBranchVectorSrc
- an2050_GSetBranchTrigMode
- an2050_GTrigger

Main

Set Trigger Mode

an2050_CSetRunTrigMode

This function sets the trigger mode.

Function Prototype:

ViStatus **an2050_CSetRunTrigMode** (ViSession **vi**, int **source**, int **polar**, ANTINT16 **trig**, int **mode**)

an2050_CSetRunTrigMode		
Parameters	**Variable Type**	
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
source	int	Specifies the source of trigger signal. 0 Trigger A input 1 Trigger B input 2 VXI Source ECL 0 3 VXI Source ECL 1
polar	int	Specifies the polarity. 0 Trigger on negative slope. 1 Trigger on positive slope.
trig	ANTINT16	Specifies the trigger modes. 0 Free Run Mode. The output waveform will start as soon as the trigger circuit is armed by the Sequencer, and will continue until stopped by the Sequencer or by the CPU with CPU_DISARM = 1. 1 Triggered Start Mode 2 Triggered Stop Mode 3 Gated by Trigger Mode 4 Triggered Start/Stop Mode
mode	int	Specifies action if the Sequencer is currently running when function is called. 0 Do not set. 1 Set even if the Sequencer is running.

<OUTPUT>		
None	None	None
<RETURN>		
None	None	None

Set Trigger Level `an2050_CSetTrigLevel`

This function sets the Trigger Level.

Function Prototype:

ViStatus **`an2050_CSetTrigLevel`** (ViSession **vi**, float **level**, int **mode**)

`an2050_CSetTrigLevel`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
level	float	Specifies the trigger level in volts - between ± 10.7 volts.
mode	int	Specifies action if the Sequencer is currently running when function is called. 0 Do not set. 1 Set, even if the Sequencer is running.
<OUTPUT>		
None	None	None
<RETURN>		
None	None	None

Advance

Set Advance Trigger Mode `an2050_GSetAdvanceTrigMode`

This function specifies the Advance Trigger mode.

Function Prototype:

ViStatus **`an2050_GSetAdvanceTrigMode`** (ViSession **vi**, ANTUINT16 **source**, ANTUINT16 **mode**)

`an2050_GSetAdvanceTrigMode`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
source	ANTUINT16	Specifies the Advance Trigger Source. 0 External SMA Source 1 Use command 'Trigger an2050_S Advance Trigger'; (i.e., source VXI register).
mode	ANTUINT16	Specifies the Advance Trigger Mode desired. 0 Polarity negative, level TTL. 1 Polarity negative, level zero. 2 Polarity positive, level TTL. 3 Polarity positive, level zero. All other values cause this function to be ignored.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if all OK.

Trigger Advance Trigger `an2050_GTrigger`

This function specifies whether this is a branch or advance trigger and also specifies how the software should react if the current "Advance or Branch Trigger Mode" source is external SMA.

Function Prototype:

ViStatus **`an2050_GTrigger`** (ViSession **vi**, int **type**, int **mode**)

`an2050_GTrigger`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
type	int	Specifies whether branch or advance trigger. 0 Branch Trigger 1 Advance Trigger
mode	int	Specifies action if Advance Trigger Mode or Branch Trigger Mode is external SMA. 0 Do not trigger, ignore this function and return error. Note that if the trigger source is by software function, this function will set the polarity to positive and the level to TTL regardless of what the current trigger mode is. 1 Trigger and reset the "Advance or Branch Trigger Mode" source from external SMA to using the software function from now on. 2 Trigger, and then retain the current "Advance or Branch Trigger Mode" source – reset the source back to external SMA, and the polarity and level as they were before the trigger.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if all OK. an2050_GEIVPARM if any input value is invalid. an2050_GEINVCMD if mode=0 and current trigger source is external. Other errors from VXI access possible.

Branch

Set Branch Vector with Markers `an2050_SSetBranchVectorMN`

This function sets the Branch Vector 'vector' to the 'ithSeg' segment of sequence 'seqID'. This causes sequence 'seqID'' to be run and sets the markers ON or OFF for each segment in the sequence, as specified in 'm1', 'm2', 'm3', 'm1Array', 'm2Array' and 'm3Array'.

In normal marker mode, when a segment has markers turned ON, the marker pulse will appear at 32 clock periods times the associated marker position into the segment, and if the segment is looped in any way, the marker pulse will appear in every loop. This function will also cause sequence 'seqID' to be loaded if it is not currently loaded. Any invalid parameter causes an error return and nothing is done. Behavior of this function varies depending on 'mode'. If the sequence is to loop continuously, it will loop from the very first segment to the last segment after it runs through the first loop from 'ithSeg' to the last segment.

Function Prototype:

ViStatus **`an2050_SSetBranchVectorMN`** (ViSession **vi**, ANTINT16 **vector**, ANTID **seqID**, ANTINT16 **ithSeg**, int **m1**, int **m2**, int **m3**, char **m1Array[]**, char **m2Array[]**, char **m3Array[]**, ANTID ***seqNew**, int **mode**)

`an2050_SSetBranchVectorMN`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
vector	ANTINT16	Specifies the vector to assign. Valid range: 0 to 15, all others are invalid and are ignored.
seqID	ANTID	Specifies the sequence involved. This parameter together with the 'ithSeg' specifies to which segment to branch..
ithSeg	ANTINT16	Specifies that the 'ithSeg' segment in sequence 'seqID' is the segment to branch to. ithSeg = 1 for the 1st segment ithSeg = 2 for the 2nd segment, and so on.
m1	int	0 Turn marker 1 OFF for all segments. 1 Turn marker 1 ON for all segments. 2 Turn marker 1 ON or OFF according to 'm1Array[]'. All other values cause 'AN2050_SEINVPARM' error return.
m2	int	0 Turn marker 2 OFF for all segments. 1 Turn marker 2 ON for all segments. 2 Turn marker 2 ON or OFF according to 'm2Array[]'. All other values cause 'AN2050_SEINVPARM' error return.
m3	int	0 Turn marker 3 OFF for all segments. 1 Turn marker 3 ON for all segments. 2 Turn marker 3 ON or OFF according to 'm3Array[]'. All other values cause 'AN2050_SEINVPARM' error return.

m1Array[]	char	This parameter is used only if 'm1' is 2, otherwise, it is ignored. When used, each element specifies whether marker 1 of the corresponding segment (i.e., 'm1Array[0]' specifies the first segment, 'm1Array[1]' specifies the second segment, and so on) should be turned ON (1) or OFF (0). Values other than 0 or 1 cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqID' (may be obtained by calling 'an2050_SGetSeqInfo()').
m2Array[]	char	This parameter is used only if 'm2' is 2, otherwise, it is ignored. When used, each element specifies whether marker 2 of the corresponding segment (i.e., 'm2Array[0]' specifies the first segment, 'm2Array[1]' specifies the second segment, and so on) should be turned ON (1) or OFF (0). Values other than 0 or 1 cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqID' (may be obtained by calling 'an2050_SGetSeqInfo()').
m3Array[]	char	This parameter is used only if 'm3' is 2, otherwise it is ignored. When used, each element specifies whether marker 2 of the corresponding segment (i.e., 'm3Array[0]' specifies the first segment, 'm3Array[1]' specifies the second segment, and so on) should be turned ON (1) or OFF (0). Values other than 0 or 1 cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqID' (may be obtained by calling 'an2050_SGetSeqInfo()').
mode	int	Bits 0 Specifies action if Sequencer is currently running. 0 Do not set, return an2050_SESQNCRRUNNING. The old branch vector remains effective. 1 Set the branch vector, disable the sequencer and load the sequence. The Sequencer will remain disabled after this function returns. Bits 1 & 2 Specifies if the old branch vector should be unloaded. 00 Do not unload the sequence for the old branch vector. 01 Unload the sequence for the old branch vector only if it is not being used by other branch vectors. 10 Unload the sequence for the old branch vector even if it is being used by other branch vectors. Note that unloading can only be done if the Sequencer is not running or, if running, bit0 of "mode" is set. Bits 4 to 7 Specifies action if the sequence is loaded already. 0 Use the loaded sequence, ignore the marker definition specified in 'm1', 'm2', 'm1Array', and 'm2Array'. 1 Use the loaded one but change the marker definition to that specified in the above variables. 2 Keep the loaded one, make a copy of it and load the new one with the new marker definition. If this is specified and 'seqID' is already loaded, the ID of the new copy of 'seqID' is output in 'seqNew'. All other bits must be 0.
<OUTPUT>		
*seqNew	ANTID	'*seqNew' is the ID of the new copy of 'seqID' if 'seqID' is already loaded, and ('mode'&0xf0==0x20). '*seqNew' is 0 if no new copy of the 'seqID' is created (i.e., 'seqID' was not loaded when this function was called). '*seqNew' is meaningful only if 'mode'&0xf0==0x20, otherwise it contains only garbage.
<RETURN>		
		0 if all OK.

Set Branch Vector `an2050_SSetBranchVector`

This function sets the Branch Vector 'vector' to the 'ithSeg' segment of sequence 'seqID'. This causes sequence 'seqID' to be loaded if it is not currently loaded. Any invalid parameter causes an error return and nothing is done. Behavior of this function varies depending on 'mode'.

This function is translated to 'SSetBranchVectorM(vi,vector,seqID,ithSeg,0,0,0,0,0,(mode|ox10))' to always use the loaded sequence (if already loaded) and set markers OFF.

Function Prototype:

ViStatus **`an2050_SSetBranchVector`** (ViSession **vi**, ANTINT16 **vector**, ANTID **seqID**, ANTINT16 **ithSeg**, int **mode**)

`an2050_SSetBranchVector`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
vector	ANTINT16	Specifies the vector to assign. Valid range: 0 to 15, all others are invalid and is ignored.
seqID	ANTID	Specifies the sequence involved. This parameter together with the 'ithSeg' specifies to which segment to branch.
ithSeg	ANTINT16	Specifies that the 'ithSeg' segment in sequence 'seqID' is the segment to branch to. ithSeg = 1 for the 1st segment ithSeg = 2 for the 2nd segment, and so on.
mode	int	**Bit 0** Specifies action if the Sequencer is currently running. 0 Do not set, return with "an2050_SESQNCRRUNNING' to indicate that the Sequencer is running. The old branch vector remains effective. 1 Set the branch vector, stop the sequencer and load the sequence. **Bit 1 & 2** Specifies whether the old branch vector should be unloaded. 00 Do not unload the sequence for the old branch vector. 01 Unload the sequence for the old branch vector only if it is not being used by other branch vectors. 10 Unload the sequence for the old branch vector even if it is being used by other branch vectors. Note that unloading can be done only if the Sequencer is not running, or, if running, if Bit 0 of 'mode' is set.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if all OK.

Set Branch Vector with Markers `an2050_SSetBranchVectorM`

Note: This function applies to Rev. 0 Hardware only.

This function sets the Branch Vector 'vector' to the 'ithSeg' segment of sequence 'seqID'. This causes sequence 'seqID'' to be run and sets the markers ON or OFF for each segment in the sequence, as specified in 'm1', 'm2', 'm1Array' and 'm2Array'. In Independent marker mode, when a segment has markers turned ON, the marker pulse will appear at 32 clock periods times the associated marker position (set through the function 'an2050_GConfigMarkers()') into the segment, and if the segment is looped in any way, the marker pulse will appear in every loop. This function will also cause sequence 'seqID' to be loaded if it is not currently loaded. Any invalid parameter causes an error return and nothing is done. Behavior of this function varies depending on 'mode'. If the sequence is to loop in any way, it will loop from the very 1st segment to the last segment after it is run through the first loop from 'ithSeg' to the last segment. This function is translated to 'SsetBranchVectorM(vi,vector,seqID,ithSeg,0,0,0,0,0,(mode|0x10))' to always use the loaded sequence (if already loaded) and set markers off.

Function Prototype:

ViStatus **`an2050_SSetBranchVectorM`** (ViSession **vi**, ANTINT16 **vector**, ANTID **seqID**, ANTINT16 **ithSeg**, int **m1**, int **m2**, char **m1Array[]**, char **m2Array[]**, ANTID ***seqNew**, int **mode**)

`an2050_SSetBranchVectorM`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
vector	ANTINT16	Specifies the vector to assign. Valid range: 0 to 15, all others are invalid and is ignored.
seqID	ANTID	Specifies the sequence involved. This parameter together with the 'ithSeg' specifies to which segment to branch..
ithSeg	ANTINT16	Specifies that the 'ithSeg' segment in sequence 'seqID' is the segment to branch to. ithSeg = 1 for the 1st segment ithSeg = 2 for the 2nd segment, and so on.
m1	int	0 Turn marker 1 OFF for all segments. 1 Turn marker 1 ON for all segments. 2 Turn marker 1 ON or OFF according to 'm1Array[]'. All other values cause 'AN2050_SEINVPARM' error return.
m2	int	0 Turn marker 2 OFF for all segments. 1 Turn marker 2 ON for all segments. 2 Turn marker 2 ON or OFF according to 'm2Array[]'. All other values cause 'AN2050_SEINVPARM' error return.

m1Array[]	char	This parameter is used only if 'm1' is 2, otherwise it is ignored. When used, each element specifies whether marker 1 of the corresponding segment (i.e., 'm1Array[0]' specifies the first segment, 'm1Array[1]' specifies the second segment, and so on) should be turned ON (1) or OFF (0). Values other than 0 or 1 cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqID' (may be obtained by calling 'an2050_SGetSeqInfo()').
m2Array[]	char	This parameter is used only if 'm2' is 2, otherwise it is ignored. When used, each element specifies whether marker 2 of the corresponding segment (i.e., 'm2Array[0]' specifies the first segment, 'm2Array[1]' specifies the second segment, and so on) should be turned ON (1) or OFF (0). Values other than 0 or 1 cause the return of 'AN2050_GEINVPARM'. The size of this array must be at least the total number of segments in the sequence 'seqID' (may be obtained by calling 'an2050_SGetSeqInfo()').
mode	int	Bits 0 Specifies action if Sequencer is currently running. 0 Do not set, return an2050_SESQNCRRUNNING. The old branch vector remains effective. 1 Set the branch vector, disable the sequencer and load the sequence. The Sequencer will remain disabled after this function returns. Bits 1 & 2 Specifies if the old branch vector should be unloaded. 00 Do not unload the sequence for the old branch vector. 01 Unload the sequence for the old branch vector only if it is not being used by other branch vectors. 10 Unload the sequence for the old branch vector even if it is being used by other branch vectors. Note that unloading can only be done if the Sequencer is not running or, if running, bit0 of "mode" is set. Bits 4 to 7 Specifies action if the sequence is loaded already. 0 Use the loaded sequence, ignore the marker definition specified in 'm1', 'm2', 'm1Array', and 'm2Array'. 1 Use the loaded one but change the marker definition to what is specified in the above mentioned variables. 2 Keep the loaded one, make a copy of it and load the new one with the new marker definition. If this is specified and 'seqID' is already loaded, the ID of the new copy of 'seqID' is output in 'seqNew'.
<OUTPUT>		
*seqNew	ANTID	'*seqNew' is the ID of the new copy of 'seqID' if 'seqID' is already loaded, and ('mode'&0xf0==0x20). '*seqNew' is 0 if no new copy of the 'seqID' is created (i.e., 'seqID' was not loaded when this function was called). '*seqNew' is meaningful only if 'mode'&0xf0==0x20, otherwise it contains only garbage.
<RETURN>		
		0 if all OK.

Set Branch Vector Source `an2050_GBranchVectorSrc`

This function specifies the source used for the selection of branch vector in Branch Memory. 'Auto Increment' causes the systematic cycling through all 16 branch vectors. For example, vector 0 is used for the 1st branch trigger, vector 1 for the 2nd branch trigger, and so on up to 16, then vector 0 again for the 17th branch trigger, and so on.

Function Prototype:

ViStatus **`an2050_GBranchVectorSrc`** (ViSession **vi**, ANTINT16 **source**)

`an2050_GBranchVectorSrc`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
source	ANTINT16	Specifies the branch vector source: 0 Front Panel Input 1 Auto-Increment
<OUTPUT>		
None	None	None
<RETURN>		
		0 if all OK.

Set Branch Trigger Mode `an2050_GSetBranchTrigMode`

This function specifies the trigger mode for branching.

Function Prototype:

ViStatus **`an2050_GSetBranchTrigMode`** (ViSession **vi**, ANTUINT16 **source**, ANTUINT16 **mode**)

`an2050_GSetBranchTrigMode`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
source	ANTUINT16	Specifies the Branch Trigger source: 0 Trigger source is external SMA 1 Use software function 'an2050_GTrigger()' to trigger.
mode	ANTUINT16	Specifies the Branch Trigger Mode desired. This is only used if the source specified is external SMA. 0 Polarity negative, level TTL. 1 Polarity negative, level zero. 2 Polarity positive, level TTL. 3 Polarity positive, level zero. All other values are ignored.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if all OK.

Trigger Branch Trigger `an2050_GTrigger`

This function specifies whether this is a branch or advance trigger and also specifies how the software should react if the current "Advance or Branch Trigger Mode" source is external SMA.

Function Prototype:

ViStatus **`an2050_GTrigger`** (ViSession **vi**, int **type**, int **mode**)

`an2050_GTrigger`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
type	int	Specifies whether branch or advance trigger. 0 Branch Trigger 1 Advance Trigger
mode	int	Specifies action if Advance Trigger Mode or Branch Trigger Mode is external SMA. 0 Do not trigger, ignore this function and return error. Note that if the trigger source is by software function, this function will set the polarity to positive and the level to TTL regardless of what the current trigger mode is. 1 Trigger, and reset the "Advance or Branch Trigger Mode" source from external SMA to using the software function from now on. 2 Trigger, and retain the current "Advance or Branch Trigger Mode" source – reset the source back to external SMA, and the polarity and level as they were before the trigger.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if all OK. an2050_GEIVPARM if any input value is invalid. an2050_GEINVCMD if mode=0 and current trigger source is external. Other errors from VXI access possible.

Marker

These functions control the marker width, mode and delays. The following functions are included:

- an2050_GSetMarker
- an2050_GSetMarkerMode
- an2050_GConfigMarkers

Set Markers — an2050_GSetMarker

Note: This function is for use with Rev 1. hardware and above. This function will return an error if used with Rev. 0 hardware.

This function sets the position and pulse width of the marker specified by 'index'.

Function Prototype:

ViStatus **an2050_GSetMarker** (ViSession **vi**, int **index**, ANTINT32 **pos**, ANTINT32 **width**)

an2050_GSetMarker		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
index	int	Specifies the marker being configured: Marker 1 Marker 2 Marker 3
pos	ANTINT32	Specifies the position of the marker output pulse relative to the start of the segment. Range: 0 to 65535. If an out of range value is entered and error will be returned. To retain the old position – enter -1.
width	ANTINT32	Specifies the width of the marker pulse. The effective pulse width will be ('width'+1)*32 sample clock periods. Range: 0 to 4095 If an out of range value is entered and error will be returned. To retain the old position – enter -1.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_GEINVPARM if invalid parameter. Other errors may appear.

Set Marker Mode `an2050_GSetMarkerMode`

Note: This function is only for use with Rev. 0 hardware.
For Rev. 1 hardware and above use the function an2050_GSetMarker.

This function is used to set the marker mode to 'mode'.

Function Prototype:

ViStatus **`an2050_GSetMarkerMode`** (ViSession **vi**, int **mode**)

`an2050_GSetMarkerMode`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
mode	int	Specifies the marker mode. 0 Independent marker mode. In this mode, the two markers act independently and each marker pulse width is 2 x 32 sampling clocks. When a segment has markers turned on, the marker1 pulse will appear at 32 times 'm1Pos' clock periods into the segment. The marker2 pulse will appear at 32 times 'm2Pos' clock periods into the segment, and if the segment is looped in any way, the marker pulse will appear in every loop. 1 Combined marker mode. In this mode, marker2 will go high at a time determined by 'm2Pos' and go low at a time determined by 'm1Pos'. Note that marker2 may stay high across segments. For example, if 'm2Pos'>'m1Pos', marker2 will go high and stay high throughout this segment and beyond until a segment with marker1 turned on is reached. If marker1 is not enabled in the combined mode, marker2 will go high and remain high indefinitely, or until the marker mode is changed to 'independent marker mode' using this command. This allows the combined mode gate pulse on marker2 to span multiple segments if desired. Marker1 behaves in the same way as it does in independent marker mode. If 'm1Pos' and 'm2Pos' are set to the same value in the combined mode, marker2 will not be activated. See 'an2050_GConfigMarkers()' description for 'm1Pos', and 'm2Pos' parameters.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_GEINVPARM if invalid parameter. Other errors may appear.

Configure Markers `an2050_GConfigMarkers`

Note: This function is only for use with Rev. 0 hardware.
For Rev. 1 hardware and above use the function an2050_GSetMarker.

This function is used to set the marker mode to 'mode' and allows setting of the marker delays.

Function Prototype:

ViStatus **`an2050_GConfigMarkers`** (ViSession **vi**, int **mode**, ANTINT32 **m1Pos**, ANTINT32 **m2Pos**)

`an2050_GConfigMarkers`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
mode	int	Specifies the marker mode. 0 Independent marker mode. In this mode, the two markers act independently and each marker pulse width is 2 x 32 sampling clocks. When a segment has markers turned on, the marker1 pulse will appear at 32 times 'm1Pos' clock periods into the segment. The marker2 pulse will appear at 32 times 'm2Pos' clock periods into the segment, and if the segment is looped in any way, the marker pulse will appear in every loop. 1 Combined marker mode. In this mode, marker2 will go high at a time determined by 'm2Pos' and go low at a time determined by 'm1Pos'. Note that marker2 may stay high across segments. For example, if 'm2Pos'>'m1Pos', marker2 will go high and stay high throughout this segment and beyond until a segment with marker1 turned on is reached. If marker1 is not enabled in the combined mode, marker2 will go high and remain high indefinitely, or until the marker mode is changed to 'independent marker mode' using this command. This allows the combined mode gate pulse on marker2 to span multiple segments if desired. Marker1 behaves in the same way as it does in independent marker mode. If 'm1Pos' and 'm2Pos' are set to the same value in the combined mode, marker2 will not be activated.
m1Pos	ANTINT32	Specifies where in the segment marker1 should appear. For any segment with marker1 turned on, marker1 pulse will appear at 32 times 'm1Pos' clock periods into the segment. Valid range is 0 to 65535. Use 1 if the old value is to be retained.
m2Pos	ANTINT32	Specifies where in the segment marker2 should appear. For any segment with marker1 turned on, marker2 pulse will appear at 32 times 'm2Pos' clock periods into the segment. Valid range is 0 to 65535. Use 1 if the old value is to be retained.
<OUTPUT>		
None	None	None
<RETURN>		
		0 if OK. an2050_GEINVPARM if invalid parameter. Other errors may appear.

Informational

These functions return informational messages to the user. The following functions are included:

- an2050_SSegsAvail
- an2050_SWaveMemAvail
- an2050_SGetSeqInfo
- an2050_SGetSeqSegs
- an2050_SGetLoadedSeqs
- an2050_SGetLoadedWaves
- an2050_SGetNumSeqsLoaded
- an2050_SGetNumWavesLoaded
- an2050_SGetState
- an2050_errorMessage
- an2050_CGetDigiBdRevNSN
- an2050_CGetOutputBdRevNSN
- an2050_CGetSNNCalibDate
- an2050_CGetCalibRevNDN
- an2050_CGetRevisionNumbers
- an2050_CGetSerialNumbers

Query Number of Segments Available — an2050_SSegsAvail

This function returns the total number of available segment data records in Sequencer Memory.

Function Prototype:

ViStatus **an2050_SSegsAvail** (ViSession **vi**, ViPUInt16 **avail**)

an2050_SSegsAvail		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
<OUTPUT>		
avail	ViPUINT16	Total number of available segment data records in Sequencer Memory.
<RETURN>		
		0 if all OK.

Query Waveform Data Memory Available `an2050_SWaveMemAvail`

This function outputs the total number of bytes of waveform data memory available.

Function Prototype:

ViStatus **`an2050_SWaveMemAvail`** (ViSession **vi**, ANTUINT32 ***avail**)

`an2050_SWaveMemAvail`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
<OUTPUT>		
*avail	ANTUINT32	Total number of bytes of waveform data memory available.
<RETURN>		
		0 if all OK.

Get Sequence Information `an2050_SGetSeqInfo`

Given the number of points per cycle desired, ‘nPts’, this function calculates the minimum number of points, ‘*tPts’ needed based on ‘WAVMaxPts’, ‘WAVMinPts’, ‘WAVMod’ of the waveform generation context specified by the ‘cntxtID’ that is associated with ‘vi’, and ‘minPts’ so that ‘*tPts’ is a multiple of ‘nPts’. If within ‘maxPts’ no total number of points can be found to satisfy these criteria, the total number of points that will give the least deviation from complete cycles of ‘nPts’ per cycle is output.

Function Prototype:

ViStatus **`an2050_SGetSeqInfo`** (ViSession **vi**, ANTID **seqID**, ANTINT16 ***numSegs**, ANTINT16 ***segRcd**, ANTINT16 ***loopMode**)

`an2050_SGetSeqInfo`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
seqID	ANTID	Specifies the ID of the sequence for which information is desired.
<OUTPUT>		
*numSegs	ANTINT16	Specifies the number of segments in the sequence. This ‘*numSegs’ can be used to allocate the proper size buffer for use in a call to ‘an2050_SGetSeqSegs()’ to get all the segment IDs in the specified sequence.
*segRcd	ANTINT16	‘*segRcd’ = -1 if sequence ‘seqID’ is not loaded. All non-negative values indicate the segment record number where the first segment of ‘seqID’ is loaded.
*loopMode	ANTINT16	Specifies loop mode. 0 If One Shot 1 If Loop Continuously
<RETURN>		
		0 if all OK. VI_ERROR_INV_SESSION: if ‘vi’ is invalid. AWSEINVID: if ‘seqID’ is invalid.

Get Segments in Sequence `an2050_SGetSeqSegs`

This function gets the segments of the sequence 'seqID'.

Function Prototype:

ViStatus **`an2050_SGetSeqSegs`** (ViSession **vi**, ANTID **seqID**, ANTINT16 ***numSegs**, ANTID **segBuf[]**)

`an2050_SGetSeqSegs`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
seqID	ANTID	Specifies the ID of the sequence whose segments are to be output.
*numSegs	ANTINT16	As input, specifies the size of the 'segBuf' array.
<OUTPUT>		
*numSegs	ANTINT16	As output, is the actual number of segments in the sequence 'seqID'. If this is greater than the input '*numSegs', only 'input *numSegs' segments is output in 'segBuf'. The caller should keep a copy of the 'input *numSegs' to check against this 'output *numSegs' to see is 'segBuf' is big enough to hold all the segments.
segBuf	ANTID	This is the buffer to receive the IDs of the segments of the sequence 'seqID'. The segments in this output array are arranged in running order.
<RETURN>		
		0 if all OK. VI_ERROR_INV_SESSION: if 'vi' is invalid. AWSEINVID: if 'seqID' is invalid.

Get Loaded Sequences `an2050_SGetLoadedSeqs`

This function gets the IDs of all loaded sequences.

The user may use an2050_SGetNumSeqsLoaded() to get the total number or sequences loaded and then allocate memory for 'seqIDs[]' and *then* call this function.

Function Prototype:

ViStatus **`an2050_SGetLoadedSeqs`** (ViSession **vi**, ANTUINT16 **nSeqs**, ANTID **seqIDs[]**, ANTUINT16 ***nSeqsOut**)

`an2050_SGetLoadedSeqs`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
nSeqs	ANTUINT16	Specifies the size of the array 'seqIDs[]'. If there are more than 'nSeqs' sequences loaded, only the IDs of the most recently defined 'nSeqs' of loaded sequences will be output in 'seqIDs[]'. If there are fewer than 'nSeqs' sequences loaded, only the first '*nSeqsOut' elements of the array 'seqIDs[]' will be valid.
<OUTPUT>		
seqIDs[]	ANTID	This buffer contains the IDs of loaded sequences. Size of this buffer must be at least 'nSeqs'.
*nSeqsOut	ANTUINT16	'*nSeqsOut' is the number of sequences loaded.
<RETURN>		
None	None	None

Get Loaded Waves `an2050_SGetLoadedWaves`

This function gets the IDs of all loaded waveforms.

The user may use an2050_SGetNumWavesLoaded() to get the total number or sequences loaded and then allocate memory for 'waveIDs[]' and *then* call this function.

Function Prototype:

ViStatus **`an2050_SGetLoadedWaves`** (ViSession **vi**, ANTUINT16 **nWaves**, ANTID **waveIDs[]**, ANTUINT16 ***nWavesOut**)

`an2050_SGetLoadedWaves`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
nWaves	ANTUINT16	Specifies the size of the array 'wavesIDs[]'. If there are more than 'nWaves' waveforms loaded, only the IDs of the most recently defined 'nWaves' of loaded waveforms will be output in 'waveIDs[]'. If there are fewer than 'nWaves' sequences loaded, only the first '*nWavesOut' elements of the array 'waveIDs[]' will be valid.
<OUTPUT>		
waveIDs[]	ANTID	This buffer contains the IDs of loaded waveforms. Size of this buffer must be at least 'nWaves'.
*nWavesOut	ANTUINT16	'*nWavesOut' is the number of waveforms loaded.
<RETURN>		
None	None	None

Get Number of Sequences Loaded `an2050_SGetNumSeqsLoaded`

This function gets the total number of sequences loaded.

Function Prototype:

ViStatus **`an2050_SGetNumSeqsLoaded`** (ViSession **vi**, ANTUINT16 ***nSeqs**)

`an2050_SGetNumSeqsLoaded`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
*nSeqs	ANTUINT16	'nSeqs' is the total number of sequences loaded.
<RETURN>		
None	None	None

Get Number of Waveforms Loaded `an2050_SGetNumWavesLoaded`

This function gets the total number of waveforms loaded.

Function Prototype:

ViStatus **`an2050_SGetNumWavesLoaded`** (ViSession **vi**, ANTUINT16 ***nWaves**)

`an2050_SGetNumWavesLoaded`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
*nWaves	ANTUINT16	'nWaves' is the total number of waveforms loaded.
<RETURN>		
None	None	None

Get State `an2050_SgetState`

This function returns the state of the sequencer. The sequencer has three states:

- running
- idling
- calibrating

Function Prototype:

ViStatus **`an2050_SGetState`** (ViSession **vi**, ANTINT32 ***state**)

`an2050_SGetState`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	Specifies the instrument handle for the selected DBS2050A.
<OUTPUT>		
*state	ANTINT32	Returns the current state of the DBS2050A. 0 Sequencer is disabled 1 Sequencer running 2 Sequencer is Idle 3 DBS2050A is in calibration process
<RETURN>		
		0 if OK.

Translate Error Code to Message `an2050_errorMessage`

This function accepts an error code as input and returns the error message corresponding to the error code.

Function Prototype:

ViStatus **`an2050_errorMessage`** (ViSession **vi**, ViStatus **errorCode**, ViPString **errorMessage**)

`an2050_errorMessage`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
errorCode	ViStatus	Specifies the error code to be translated.
<OUTPUT>		
errorMessage	ViPString	This output buffer contains the error message corresponding to the input error code. If the input error code cannot be matched against any known values, then this parameter contains the value "Unknown Error Code: 0xnnnnnnnn" where 'nnnnnnnn' is the error code in hexadecimal. The size of 'errorMessage' must be at least 512 bytes wide.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_WARN_UNKNOWN_STATUS': 'errorCode' is not known by the driver.

Get Digital Board Revision & Serial Number `an2050_CGetDigiBdRevNSN`

This function outputs the Digital Board Revision number in 'rev' and the Serial number in 'SN'. 'rev' and 'SN' must each be at least 16 bytes.

Function Prototype:

ViStatus **`an2050_CGetDigiBdRevNSN`** (ViSession **vi**, ViChar _VI_FAR **rev[]**, ViChar _VI_FAR **SN[]**)

`an2050_CGetDigiBdRevNSN`		
Parameters		**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
rev[]	ViChar _VI_FAR	The Digital Board Revision number is output in 'rev' as an ASCII string. 'rev' must be at least 16 bytes.
SN[]	ViChar _VI_FAR	The Digital Board Serial number is output in 'SN' as an ASCII string. 'SN' must be at least 16 bytes.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_NSUP_OPER': "Unsupported operation." If the hardware revision is smaller than 2 hence no digital board revision or digital board serial number is recorded in EEPROM.

Get Output Board Revision & Serial Number `an2050_CGetOutputBdRevNSN`

This function outputs the Output Board Revision number in 'rev' and the Serial number in 'SN'. 'rev' and 'SN' must each be at least 16 bytes.

Function Prototype:

ViStatus **`an2050_CGetOutputBdRevNSN`** (ViSession **vi**, ViChar _VI_FAR **rev[]**, ViChar _VI_FAR **SN[]**)

`an2050_CGetOutputBdRevNSN`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
rev[]	ViChar _VI_FAR	The Output Board Revision number is output in 'rev' as an ASCII string. 'rev' must be at least 16 bytes.
SN[]	ViChar _VI_FAR	The Output Board Serial number is output in 'SN' as an ASCII string. 'SN' must be at least 16 bytes.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_NSUP_OPER': "Unsupported operation." If the hardware revision is smaller than 2 hence no output board revision or output board serial number is recorded in EEPROM.

Get DBS2050A Serial Number & Calibration Date `an2050_CGetSNNCalibDate`

This function outputs the DBS2050A Serial Number in 'SN' and the Calibration Date in 'date'. 'SN' and 'date' must each be at least 16 bytes.

Function Prototype:

ViStatus **`an2050_CGetSNNCalibDate`** (ViSession **vi**, ViChar _VI_FAR **SN[]**, ViChar _VI_FAR **date[]**)

`an2050_CGetSNNCalibDate`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
SN[]	ViChar _VI_FAR	The DBS2050A Serial number is output in 'SN' as an ASCII string. 'SN' must be at least 16 bytes.
date[]	ViChar _VI_FAR	The Calibration Date is output in 'date' as an ASCII string. 'date' must be at least 16 bytes.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_NSUP_OPER': "Unsupported operation." If the hardware revision is smaller than 2 hence no DBS2050A Serial Number or calibration date is recorded in EEPROM.

Get Calibration Procedure Rev. and Document # `an2050_CGetCalibRevNDN`

This function outputs the Calibration Procedure Revision number in 'rev' and the document number in 'DN'. 'rev' and 'DN' each must be at least 16 bytes large.

Function Prototype:

ViStatus **`an2050_CGetCalibRevNDN`** (ViSession **vi**, ViChar _VI_FAR **rev[]**, ViChar _VI_FAR **DN[]**)

`an2050_CGetCalibRevNDN`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
rev[]	ViChar _VI_FAR	The Calibration Procedure Revision number is output in 'rev' as an ASCII string. 'rev' must be at least 16 bytes.
DN[]	ViChar _VI_FAR	The Calibration Procedure Document number is output in 'DN' as an ASCII string. 'DN' must be at least 16 bytes.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_NSUP_OPER': "Unsupported operation." If the hardware revision is smaller than 2 hence no calibration revision or document number is recorded in EEPROM.

Get Digital, Output, Calibration Revision #s `an2050_CGetRevisionNumbers`

This function outputs:

- Digital Board Revision number in 'digiRev'
- Output Board Revision number in 'outRev'
- Calibration Procedure Revision number in 'calibRev'

Each of these buffers must be at least 16 bytes large.

Function Prototype:

ViStatus **`an2050_CGetRevisionNumbers`** (ViSession **vi**, ViChar _VI_FAR **digiRev[]**, ViChar _VI_FAR **outRev[]**, ViChar _VI_FAR **calibRev[]**)

`an2050_CGetRevisionNumbers`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
digiRev[]	ViChar _VI_FAR	The Digital Board Revision number is output in 'digiRev' as an ASCII string. 'digiRev' must be at least 16 bytes.
outRev[]	ViChar _VI_FAR	The Output Board Revision number is output in 'outRev' as an ASCII string. 'outRev' must be at least 16 bytes.
calibRev[]	ViChar _VI_FAR	The Calibration Procedure Revision number is output in 'calibRev' as an ASCII string. 'calibRev' must be at least 16 bytes.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_NSUP_OPER': "Unsupported operation." If the hardware revision is smaller than 2 then no revision number is recorded in EEPROM.

Get Digital and Output Board Serial #s `an2050_CGetSerialNumbers`
Get Calibration Procedure Document Number

This function outputs:

- Digital Board Serial number in 'digiSer'
- Output Board Serial number in 'outSer'
- Calibration Procedure Document number in 'calibDoc'

Each of these buffers must be at least 16 bytes.

Function Prototype:

ViStatus **`an2050_CGetSerialNumbers`** (ViSession **vi**, ViChar _VI_FAR **digiSer[]**, ViChar _VI_FAR **outSer[]**, ViChar _VI_FAR **calibDoc[]**)

`an2050_CGetSerialNumbers`		
Parameters	**Variable Type**	**Description**
<INPUT>		
vi	ViSession	The session of the instrument being acted upon. This was obtained by calling 'an2050_init()'.
<OUTPUT>		
digiSer[]	ViChar _VI_FAR	The Digital Board Serial number is output in 'digiSer' as an ASCII string. 'digiSer' must be at least 16 bytes.
outSer[]	ViChar _VI_FAR	The Output Board Serial number is output in 'outSer' as an ASCII string. 'outSer' must be at least 16 bytes.
calibDoc[]	ViChar _VI_FAR	The Calibration Procedure Document number is output in 'calibDoc' as an ASCII string. 'calibDoc' must be at least 16 bytes.
<RETURN>		
		ViStatus status: 'VI_SUCCESS' if successful. 'VI_ERROR_NSUP_OPER': "Unsupported operation." If the hardware revision is smaller than 2 then no serial number is recorded in EEPROM.

Appendix A
DBS2050A Performance Specifications

Note: All specifications with 50 ohm load unless otherwise specified.

Clock and Timing

Title	Specification	Notes
Maximum Sample Rate	2.4Gs/s Single Channel Mode 1.2Gs/s Dual Channel Mode	
Minimum Sample Rate	600s/s Single Channel Mode 300s/s Dual Channel Mode	
Clock Modifier (Time Base Divider)	÷1, 2, 4, 8; (High Speed) ÷8 to 64 in steps of 8; (DIVA) ÷64 to 4194304 in steps of 64; (DIVB)	Internal & External clock
Clock Resolution	≤ 0.4% of desired frequency	
Internal clock accuracy	± 2PPM typ., ± 5PPM max.	Requires 15 minute warm up after selecting internal clock to achieve specified accuracy
Clock Jitter Measurement Window 100 ns	<10 ps RMS, Hi speed and DIVA divider <20 ps RMS, DIVB divider	Internal clock only
External Reference Clock Frequency Input Range	2.5MHz to 100MHz in 2.5MHz ±1% steps	Single ended input
External Reference Clock Input Impedance	AC coupled to 50Ω	
External Reference Clock Input Voltage Range	.8Vp-p to 2Vp-p .5Vp-p to 2Vp-p	Sine @ 10MHz Square wave Slew Rate ≥ .5V/ns
External Clock Frequency Range	100kHz to 2.4 GHz	Single ended input
External Clock Input Impedance	AC coupled to 50Ω	
External Clock Input Voltage Range	.8Vp-p to 2Vp-p	Min. Slew Rate ≥ .5V/ns

Output Signal Characteristics

Title	Specification	Notes
Output mode configurations	Single Channel Mode: Channel 1: differential X1 Channel 1: single ended X1 Channel 1: single ended X4 Dual Channel Mode: Channel 1: single ended X1, X4 Channel 2: single ended X1, X4	Max clock = 2.4Gs/s Max clock = 1.2Gs/s
Waveform Vertical resolution	8 bit	
Max Waveform Amplitude	Single ended X1: 1 Volt pk-pk Differential X1: 2 Volt pk-pk Single ended X4: 4 Volt pk-pk	
Low Frequency Waveform Amplitude accuracy	±2% of setting at max output	
Waveform Integral non-linearity	1% of FSR	Best Straight Line
Waveform Differential non-linearity	0.22% of FSR	
Gain Control Range	0.001 to 1.00 in approx. 4000 steps	
Gain Control accuracy	±1% of Gain, ±0.00025V/V	
DC Offset range	-3.5 to 3.5V in approx. 2mV steps -2 to +2 in approx. 1mV steps	Single ended output and Common Mode for Differential output. Differential mode output
Offset Accuracy of Single Ended X1	±2% of Offset ±10mV ±1% of Gain Setting	
Offset Accuracy of Single Ended X4	±2% of Offset ±40mV ±4% of Gain Setting	
Offset Accuracy of Differential output: Common Mode offset Differential Offset	Single ended/differential X1: ±2% of Offset ±10mV ±1% of Gain Setting ±2% of Offset ±10mV ±2% of Gain Setting	Common Mode Offset = 0V

Output Signal AC Specifications

Title	Specification	Notes
Rise/Fall time (10% to 90%)	Single ended/differential X1: <500ps, up to 0.5Vp-p <600ps, 0.5 to 1Vp-p Single ended X4: <2.2ns up to 2Vp-p <2.5ns up to 4Vp-p	
Overshoot	Single ended/differential X1: < 7% Single ended X4: <7%	
Settling time	Single ended X1: <50ns to within 2% of step size Single ended X4: <50nS to within 2.2% of step size	Full scale step
Output filters Rise Time (3dB bandwidth)	175nS ±15% (2MHz) 17.5nS ±15% (20MHz) 1.75nS ±15% (200MHz)	3 pole Bessel BW = 0.35/Rise time
Output Bandwidth (3dB) Single Channel Mode:	Single ended/differential X1: >700MHz, up to 1Vp-p Single ended X4: >165MHz, up to 2Vp-p	 Corrected for Sin x/x roll off.
Sine wave Amplitude flatness Single Channel Mode:	Single ended/differential X1: ±0.5dB, DC to 30MHz ±1.5dB, 30MHz to 100MHz ±2dB, 100MHz to 300MHz +2, -3dB 300MHz to 700MHz	 Output: 1Vp-p sine for DC to 300MHz. Output: .5Vp-p sine for 300MHz to 700MHz. Clock = 2.4GHz Corrected for Sin x/x roll off.
Sine wave Amplitude flatness Dual Channel Mode:	Single ended X1: ±0.5dB, DC to 30MHz ±1.5dB, 30MHz to 300MHz +2, -3dB 300MHz to 500MHz	 Output: 1Vp-p sine for DC to 300MHz. Output: .5Vp-p sine for 300MHz to 500MHz. Clock = 1.2GHz Corrected for Sin x/x roll off.
Sine wave amplitude flatness Dual Channel Mode:	Single ended X4: ±0.5dB, DC to 30MHz ±2 B, 30MHz to 100MHz +2, -3dB 100MHz to 165MHz.	Output: 4Vp-p sine for DC to 100MHz. Output: 2Vp-p sine for 100MHz to 165MHz. Clock freq. = 1.2GHz Corrected for Sin x/x roll off.
Spurious free dynamic range of Single Channel Mode: Channel 1+ and Ch 1-	Single ended X1: >45dBc, <10MHz >30dBc, <200KHz (DC Offset>±2V) >40dBc , 10MHz to 50MHz >30dBc, 50MHz to 200MHz >25dBc, 200MHz to 400MHz	Output: 1Vp-p sine Measurement BW: 1MHz to 1.19GHz. Clock freq. = 2.4GHz.

Spurious free dynamic range Dual Channel Mode:	Single ended X1: >45dBc, <10MHz >30dBc, <200KHz (DC Offset >±2V) >35dBc, 10MHz to 50MHz >29dBc, 50MHz to 200MHz >25dBc, 200MHz to 400MHz	Output: 1Vp-p sine Measurement BW: 1MHz to 605MHz. Clock freq. = 1.2GHz
Spurious free dynamic range Dual Channel Mode:	Single ended X4: DC Offset=0V >40dBc, <10MHz >39dBc, 10 MHz to 50 MHz DC Offset <±2V >35dBc, <10MHz >28dBc, 10MHz to 50MHz DC Offset<±3.5V >35dBc, <10MHz >23dBc, 10MHz to 50MHz	Output = 4Vp-p sine Offset = 0V Measurement BW: 1MHz to 605MHz Clock freq. = 1.2GHz
Nyquist spur rejection Single Channel Mode:	Single ended X1: >40dBc at 1.2GHz	Output: 1Vp-p sine Signal: 50MHz Clock freq. = 2.4GHz
SINAD = S/(N+D): Single Channel Mode: Channel 1+ and Ch 1- Dual Channel Mode: Dual Channel Mode: S = Signal Power N+D=Noise power+Total Distortion	 Single ended X1: >37dBc, with 1Vp-p signal >30dBc, with 100mVp-p signal >14dBc, with 10mVp-p signal Single ended X1: >34dBc, with 1Vp-p signal >30dBc, with 100mVp-p signal >15dBc, with 10mVp-p signal Single ended X4: >37dBc, with 4Vp-p signal >33dBc, with 400mVp-p signal >18dBc, with 40mVp-p signal	Signal: 50MHz sine Measurement BW: 5MHz to 1.190GHz Clock freq. = 2.4GHz Measurement BW: 5MHz to 605MHz Clock freq. = 1.2GHz Measurement BW: 5MHz to 605MHz Clock freq. = 1.2GHz
Channel Crosstalk	<-40dB	Output: 1Vp-p Signal: 600MHz Square
Output Impedance	50Ω ±2%	At DC
Output Protection	Output will withstand short circuit to ground indefinitely.	

Trigger

Title	Specification	Notes
Trigger Sources	Trigger A, Trigger B, VXI Trigger ECL 0 and ECL 1	
Trigger Range	±10V	
Trigger Threshold Accuracy	±5% of setting ±140mV	
Trigger Hysteresis	40mV typical	
Trigger maximum safe input	±23V continuously	
Trigger input impedance	4k ±5%	
Triggering modes	Free run, trig start, trig stop, trig start/stop, gated	
Trigger to output delay	Single Channel Mode: 28nsec ±14nsec +35/Fs + Trigger delay uncertainty Dual Channel Mode: 28nsec ±14nsec +17/Fs + Trigger delay uncertainty	Fs = Sample Rate
Trigger delay uncertainty	840pS (fs=2.4GHz) Worst case any clock 1.7mS	

Trigger delay Adjustment	>2nsec nominal range (1LSB=10.5ps typ) Trigger delay adjust allows elimination of synchronous trigger delay uncertainty.	With External Clock only and not using Clock Synch mode. Input Trigger to Clock delay must be held constant.
Trigger Hold-off	1usec + (N+32)/Fs; N ≥ 512	N = Number samples Fs = Sample Rate
Branch Trigger delay	1.66usec + 256/Fs + Branch Trigger uncertainty	Fs = Sample Rate
Branch Trigger delay uncertainty	(134nsec+32/Fs) typ.	Fs = Sample Rate

Waveform (Segment, Sequencing, Branching, Amplitude, MARKERs)

Title	Specification	Notes
Waveform segments	4096	
Waveform memory depth	8 MB	4 MB per channel in dual channel mode
Waveform Branching	Yes	Under the external control of 16 codes (4 bits), the instrument can branch to output different waveform segments.
Waveform modulus	32 samples in single channel mode 16 samples in dual channel mode	
Amplitude modulation pattern control	Both Gain and Offset can be modulated	Signal amplitude for channels 1 and 2 can be updated to new amplitude settings for each waveform segment.
Amplitude control settling time Offset control settling time	50ns to within 10% of gain change. 200ns to within 2% of gain change. 6us to within 2% of Offset change.	Measured from minimum to maximum gain step
Trigger branch on “flag”	Yes	As described above in waveform branching
MARKER Output Drive	TTL output	
MARKER Output Impedance	50Ω	
MARKER 1, 2, 3 Delay	Single Channel Mode: Marker Delay = N * (32/Fs); Dual Channel Mode: Marker Delay = N * (16/Fs); N = 0 to (2^{16})-1 Fs = sample frequency	Delay begins at the start of the waveform sequence
MARKER 1, 2, 3 Pulse Width	Single Channel Mode: Marker Width = (M+1) * (32/Fs) Dual Channel Mode: Marker Width = (M+1) * (16/Fs) M = 0 to (2^{12})-1	Fs = sample frequency
Branch Vector Input Voltage Range	VIL <= +.8V, VIH => +2V	
Branch Trigger Input Voltage Threshold	V threshold = 0V, +1.5V programmable	
Branch Trigger Input Safe Operating Range	±23V continuous (Input voltage range is ±10V)	
Advance Trigger Input Voltage Threshold	V threshold = 0V, +1.5V programmable	
Advance Trigger Input Safe Operating Range	±23V continuous (Input voltage range is ±10V)	

Calibration

Title	Specification	Notes
Calibration/adjustment	Once per 12 months.	Calibration factors stored in Flash RAM.

Software

Title	Specification	Notes
Waveform editing/creation tools	VXIplug&play Driver	With Soft Front panel
Load waveform from file	VXIplug&play Driver	With Soft Front panel
High level waveform control language	VXIplug&play Driver	
Simulator support	Inquire Factory	
LabView Support	VXIplug&play Driver	
LabWindows CVI support	VXIplug&play Driver	

VXI

Title	Specification	Notes
HW Interface	VXI Compliant: A16/A24/A32, D16/D32 A24/A32 Mode Jumper	Factory Default jumper setting: A32
Interface speed	VXI	Optimized design supporting D32 and Block transfers
Data format	4 Bytes per write in D32 Mode 2 Bytes per write in D16 Mode	
VXI Logical Address Range	0h to 254h; Address 255h is reserved	Factory Default Logical Address: 128

Environmental

Title	Specification	Standard
Operating Temp	10°C to 40°C	IEC 68-2-1 (Ad) IEC 68-2-2 (Bd)
Operating Humidity	40°C @ 95% RH non-condensing	IEC 68-2-3 (Ca) Part 2
Non-Operating Temp and Humidity	-20°C @ 5% RH to +70°C @ 95% RH Damp Heat, Cyclic	IEC 68-2-1 (Ab) IEC 68-2-2 (Bb) IEC 68-2-30 (Db)
Non-Operating Altitude	0 to 10,000ft. 1,800 fpm @ 40°C	IEC 68-2-13 Test M

Non-operational Shock & Vibration

Title	Specification	Standard
Sinusoidal Vibration [1]	0.15mm disp. (10Hz to 58Hz) <2G's (58Hz to 150Hz)	IEC68-2-6 (Fc)
Bump[2]	10g, half sine, 6ms to 10ms duration	IEC68-2-29 (Eb)
Transport Vibration	Two 53 minute vibratory impacts at 270CPM at 4.5Hz	ISTA Project 2A
Transport Drop	Ten 32in free fall drops	ISTA Project 2A
Shock		IEC 68-2-27 (Ea)
Random Vibration		IEC 68-2-36 (Fdb)
Bench Handling		MIL-PRF-28800

Test conditions:
1. 10 sweeps/axis, 3 Axis, 1 octave/min. sweep, 10min. dwell at resonance.
2. 1000 times on each of 3 axis.

MTBF Parts Reliability

Title	Specification	Standard
MTBF	46,000hrs. @ 25°C	MIL-HDBK-217F

EMC Immunity

Title	Specification	Test condition	Class/ Level
ESD Immunity	EN61000-4-2:1995	4kV/4kV (Air/Contact)	A/2
Radiated RF Electromagnetic Immunity	EN61000-4-3:1995	3V/M	A/2
Transient/Burst Immunity	EN61000-4-4:1995	1kV AC lines .5kV Sig. lines > 3M	A/1
Surge Immunity	EN61000-4-5:1995	5kV/1kV line-line/line-earth	A/1
Conductive Immunity	EN61000-4-6:1996	3V Power lines 3 V Sig. lines > 3M	A/1
Voltage Variation Immunity	EN61000-4-11:1995	1 cycle/100% AC lines	A

Conducted Emissions

Title	Specification	Test condition	Class/ Level
EMC compatibility for harmonic emissions	EN61000-3-2:1995	40dB(μV/M)@10M 30MHz-230MHz 47dB(μV/M)@10M 230MHz-1000MHz	A
EMC compatibility Voltage fluctuations	EN61000-3-3:1995		A

Radiated Emissions

Title	Specification	Test condition	Class/ Level
Radiated Emissions	EN55011		
Radiated Emissions	FCC Part 15		
Radiated Emissions	CSA ICES-003		

Safety

Title	Specification	Notes
Safety Analysis	EN61010-1	2nd amendment

Power

Title	Specification	Dynamic Module Current
Voltages and currents	+24V ±5% @ .10A max. -24V ±5% @ .10A max. +12V ±5% @ 0.7A max. -12V ±5% @ 1.0A max. +5.0V ±5% @ 6.2A max. +5V STDBY @ 0.0A max. -5.2V ±5% @ 8.5A max. -2V ±5% @ 3.6A max.	+24V @ .1A$_{pk-pk}$ max. -24V @ .1A$_{pk-pk}$ max. +12V @ .2A$_{pk-pk}$ max. -12V @ .2A$_{pk-pk}$ max. +5.0V @ .5A$_{pk-pk}$ max. +5V STDBY @ 0A$_{pk-pk}$ max. -5.2V @ .4A$_{pk-pk}$ max. -2V @ .3A$_{pk-pk}$ max.

Title	Specification	Notes
Fuse Capacity	Output Card: ±24V: 1A, 125VAC ±12V: 2A, 125VAC +5V: 2A, 125VAC -5.2V: 10A, 125VAC -2V: 3A, 125VAC Digital Card: +5V: 5A, 125VAC -5.2V: 5A x 2, 125VAC	No User serviceable parts. Fuses must be changed at the factory.
Power consumption	125W max.	

Physical

Title	Specification	Notes
Physical dimensions	Double Wide, VXI-C size module	
Weight	2.7 kg (6 lb)	
Cooling	4.0Ltrs/sec/slot min @ 0.5mm H_2O	

Appendix B
Pre-compensation Option for DBS2050A

Overview

Pre-compensation increases the performance of the DBS2050A by allowing creation of higher fidelity waveforms. For band-limited signals operating in characterized test systems, the option produces near perfect frequency response characteristics:

- Flatness <0.3dB from DC to 300MHz

Any DBS2050A can be configured for use with Pre-compensation. This is done at the factory during initial production or upon return of the instrument.

Pre-compensation of a waveform may be accomplished via

- the DBS2050A/2055 Soft Front Panel (SFP)
- an application program written by the user

Both methods rely on a Pre-compensation Library (.dll), and Pre-compensation Correction Data for the DBS2050A that is to run the waveform.

Contents

The pre-compensation kit includes:

1. Two Pre-compensation filter cable assemblies.
2. Floppy disk containing pre-compensation correction data for the specific DBS2050A referenced on the floppy label by the DBS2050A Serial Number. The information contained on this disk is matched to the instrument and the reference cables.

   At the factory, prior to shipment, pre-compensation correction data is calculated for a specific DBS2050A instrument and pre-compensation filter cable attached. This correction data is used to increase the accuracy (increase the frequency response flatness) of the DBS2050A to a maximum frequency of 1.19GHz ($0.5f_s$, where f_s = sample clock).
3. Pre-compensation Library (.dll) – supplied on the standard DBS2050A/2055 Software CD supplied with the DBS2050A.

Visually inspect the cable assemblies regularly for wear and tear prior to use.

Installation

Software Components

Pre-compensation Library

The Pre-compensation Library is supplied as a dynamic link library file (.dll) located on the standard DBS2050A/2055 Software CD.

The driver is automatically installed, via a simple setup.exe script, along with all other DBS2050A/2055 software.

After installation, the driver is accessible through any user developed application, or through the soft front panel provided. Refer to the Pre-compensating a Waveform Using the Soft Front Panel section of this Appendix for a description of the pre-compensation process.

Pre-compensation Correction Data from Floppy Disk

The Pre-compensation Correction Data file contained on the supplied floppy disk is unique to the referenced DBS2050A Serial Number.

Filename coding example: `2050 TDS820 Ser Num A143282.txt`

This indicates that the data supports pre-compensation of a DBS2050A Serial Number 143282. The data was obtained using the characteristic response of Model TDS820 DSO.

Note: It is recommended that this file be saved onto the working computer, and the floppy be placed into secure storage. If you lose your correction data, Analogic maintains a database of pre-compensation data on all production instruments.

- Create a folder in C:/DBS2050A_2055 called ‘Pre-comp Data’.
- Copy the correction data .txt file from the floppy disk into this folder.
- Browse to this .txt file when prompted by the Soft Front Panel or customer generated application.

Hardware Installation

Note: The two Pre-compensation Filter Cable Assemblies be used to output pre-compensated waveforms.

The cable with filter marked ‘Output 1A’ is connected to the DBS2050A Front Panel Output 1A SMA and the one marked ‘Output 2A’ is connected to Output 2A SMA. Refer to the figure below. Note that the non-filtered ends of the cable assemblies are connected to the instrument.

Note: The filtered ends must be connected to a 50 Ohm load.

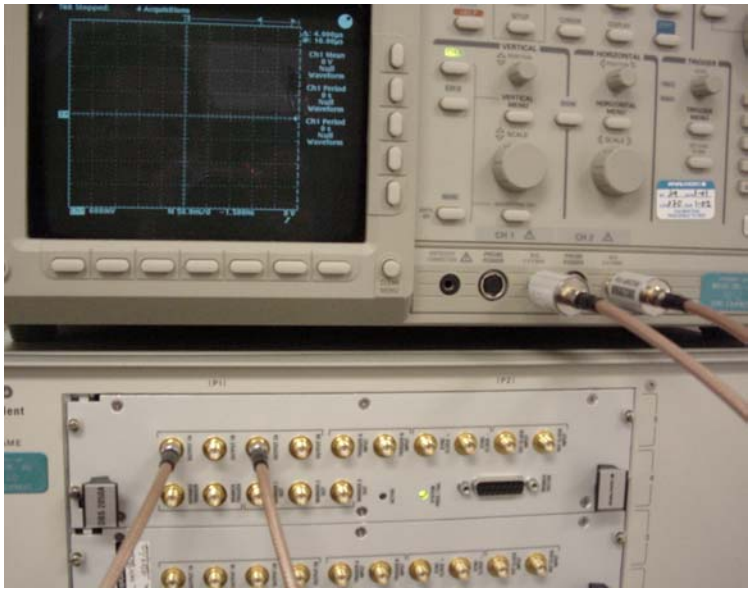
Single-Ended Output Mode	Use 1 Pre-comp Filter cable, connected to Output 1A. 
Differential Output Mode	Use both Pre-comp Filter cables, connected to Output 1A and 2A. 

Figure 17: Cable Connections for Single Channel Mode Pre-compensated Wave Output

Pre-compensation Theory

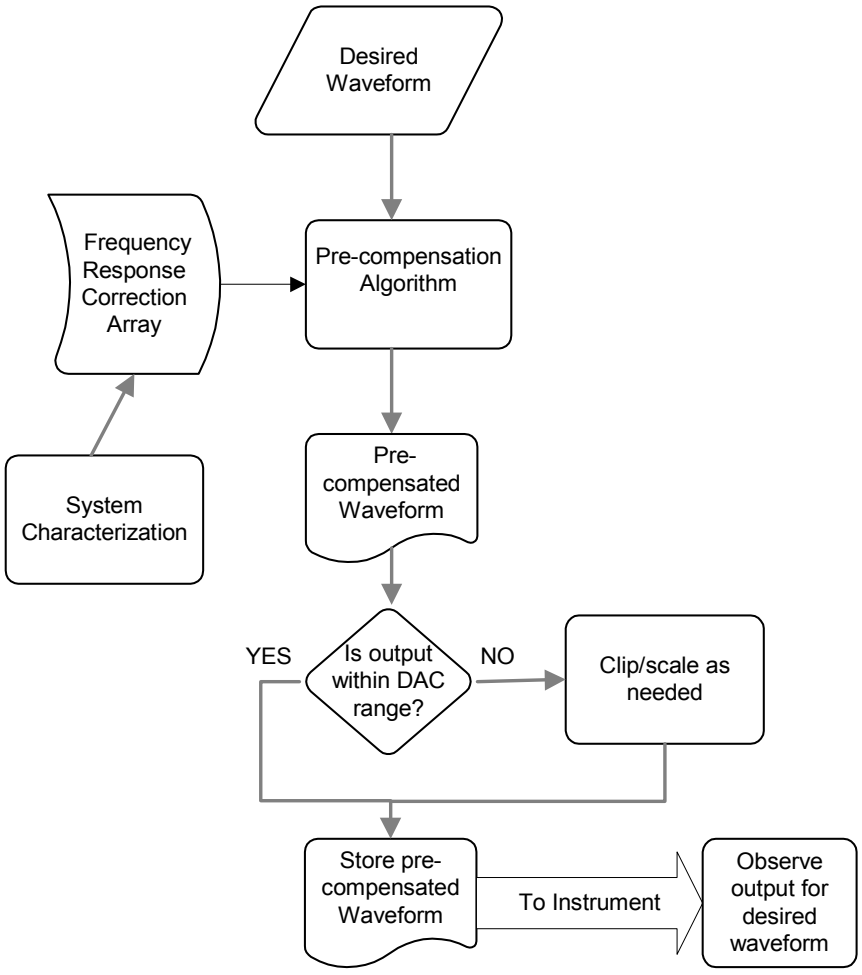
Pre-compensation Library

The Pre-compensation Library is a .dll on the standard DBS2050A/2055 VXIplug&play Software CD.

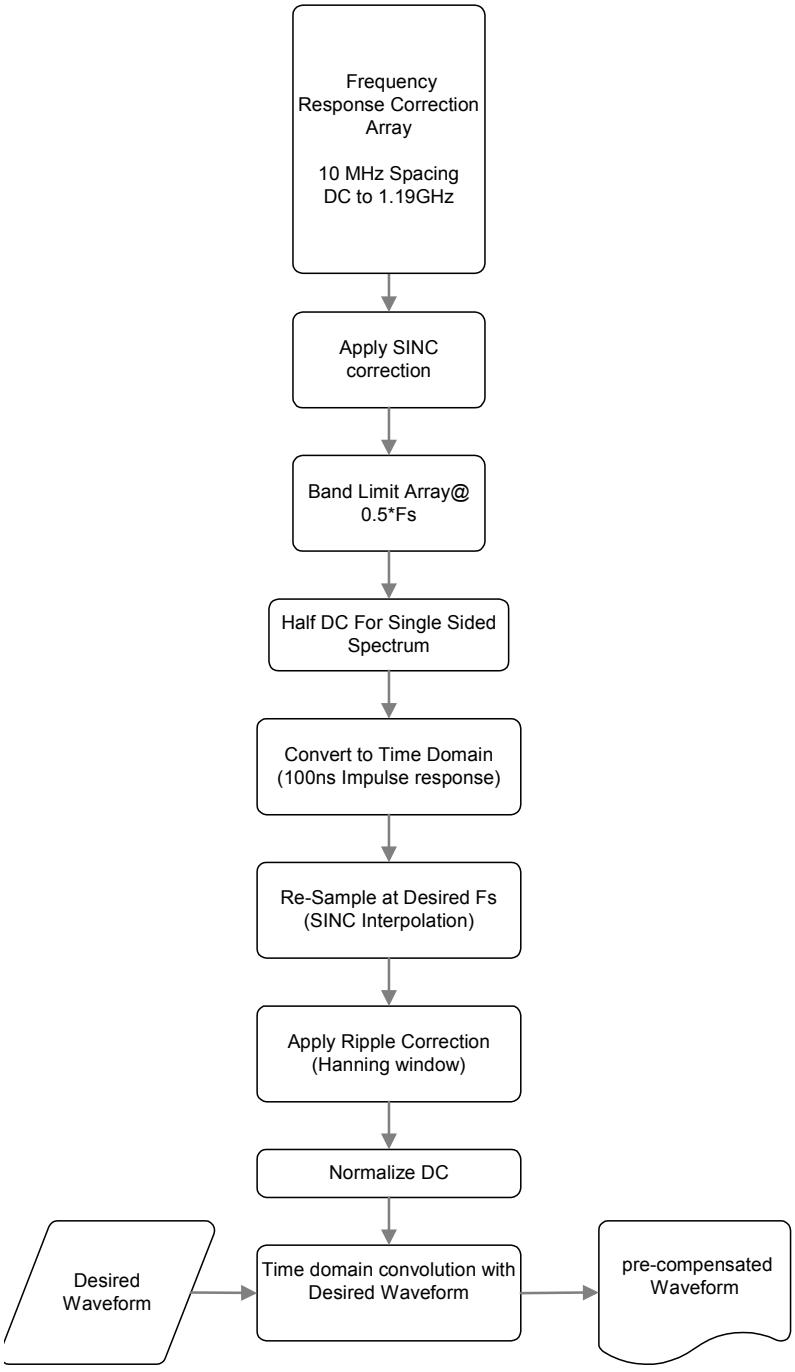
Theory of Operation – The Process of Pre-compensation

The Pre-compensation process involves a series of operations defined by an algorithm.

Pre-compensation Process Flow



Pre-compensation Algorithm



Frequency Response Correction Input – Pre-compensation Correction Data Array

The Frequency Response Correction input array contains complex data points. This array is originated by a Pre-compensation Utility program at the factory and shipped to the customer on floppy disk. It is the reciprocal of the Frequency Response of the analog section of the DBS2050A, its output cable and filter. Each data point represents a sampling in the frequency domain starting at DC with a separation as defined by the Input Freq Spacing (typically 10MHz). The last point of the array is for a frequency of 1190MHz for the DBS2050A. The array can be larger as it is truncated within the program.

SINC Correction

The DAC in the DBS2050A will cause a roll off in the frequency domain having a SINC (Sin x/x) shape, with the first zero being at the sampling frequency for a DBS2050A.

Thus each element of the Freq Resp Correction array is divided by SINC[π*Fn/Fs].

Where:
Fn is frequency corresponding to the nth element.
Fs is the sample frequency in the case of the DBS2050A

Band Limit Array

The array is resized to band limit the correction frequency to Nyquist (Fs/2) for the DBS2050A.

If the Freq Resp Correction array length is too small to reach these frequency limits then the array is padded with zeros up the limit.

Half DC

The DC component (first element) of the array is halved so that the new array represents the single-sided spectrum of the correction frequency response.

Convert to Time Domain

The resulting array is converted to the time domain, which represents the impulse response of the required correction. The width of the impulse response is 1/(Input Freq Spacing), i.e., 100ns for 10MHz spacing. The Impulse response is rotated to put Time zero in the center. The conversion is designed to provide an odd number of samples.

Re-Sampling

The impulse response is re-sampled at the Sample Frequency using a SINC interpolation algorithm. The output samples are odd and centered about the original time zero element, i.e. time zero value in equals time zero value out.

SINC interpolation is used to ensure a flat frequency response of the interpolation.

Hanning Window

A Hanning window is applied to the re-sampled impulse response. This reduces rippling in the frequency domain that would occur if the impulse response were abruptly truncated.

Normalize DC

All the elements are summed and each element divided by this sum. This normalizes the eventual output level after the upcoming convolution; i.e., after convolution, the DC output is equal to the DC input of the Desired Waveform.

Time Domain Convolution

The resulting impulse response is convolved with the Desired Waveform.

The output is now longer than the Desired Waveform thus:

Output length= Desired Waveform length + 2 * Pad size.

Where: Pad size = INT[Width of re-sampled impulse response /2]

'Pad size' number of samples (array elements) are added prior to time zero of the Desired Waveform and also to the end.

Pad to Loop Transform

If the LoopOrPad input is 1, the Pad values are wrapped and added into the array such as to produce the same size output as the size of the Desired Waveform input.

Pre-compensating a Waveform Using the Soft Front Panel

Pre-compensation is used to compensate for roll-off at higher frequencies. It should be used to ensure that the output waveform is very accurate with respect to the desired waveform. Pre-compensation may be performed via the Waveforms Panel.

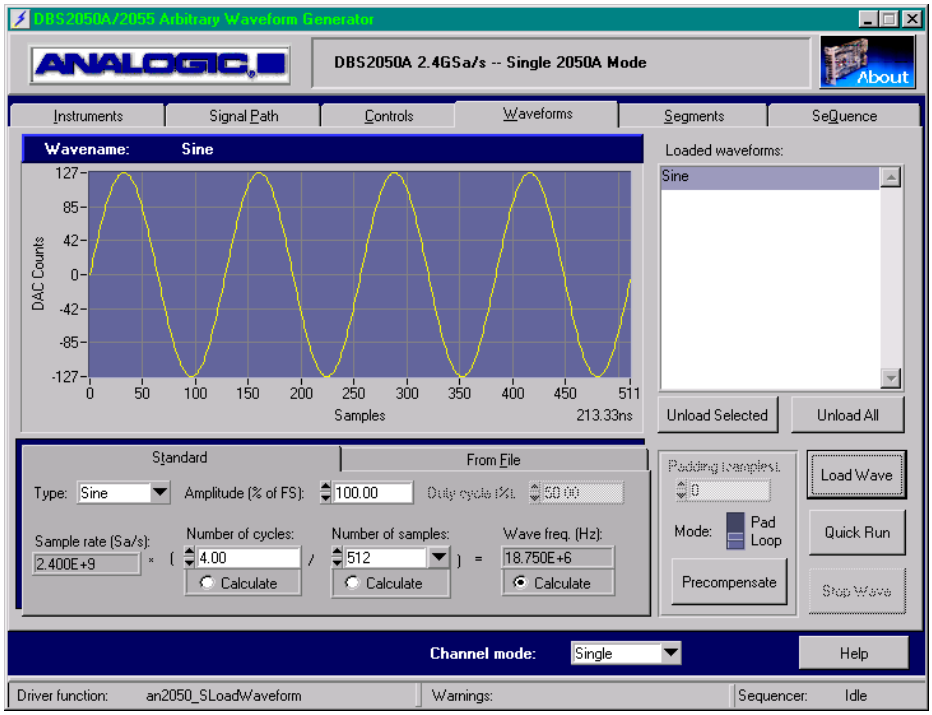


Figure 18: Waveforms Panel

To pre-compensate a waveform:

1. Using the switch labeled *Mode,* select to pre-compensate in either *Pad Mode* or *Loop Mode*.

   If *Pad Mode* is selected, enter the number of pad samples desired. This value must be modulo 32.

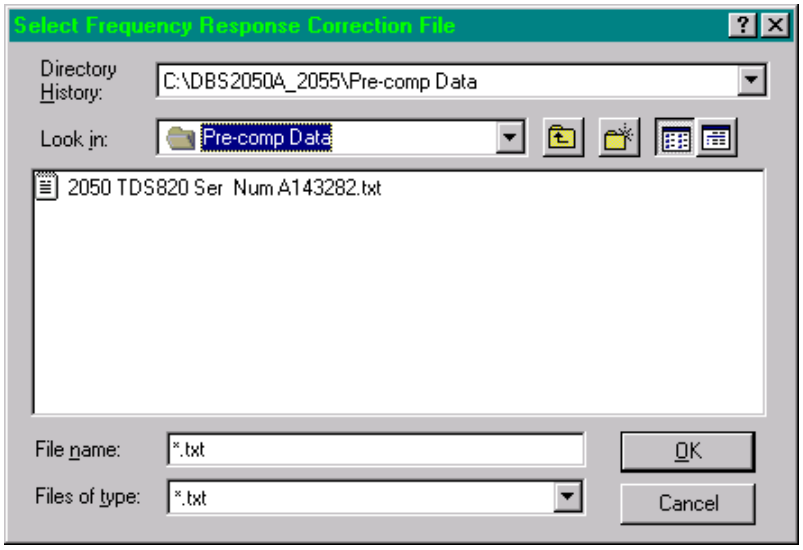
   Loop Mode implies that the wave being compensated will repeat indefinitely.

   Pad Mode implies that the wave will be played once (either in a sequence with other waves or just by itself). In Pad Mode the pre-compensation algorithm adds pad samples of value=0 to the beginning and end of the desired wave.

2. Click the *Pre-compensate* button.

   A popup browse window will appear requesting the location of the frequency response correction file. This file contains frequency response correction data generated at the factory for this specific

DBS2050A and is shipped with the instrument if the option is purchased.
Note: The frequency response correction file contains the data for both single and differential output configurations. Hence, the output configuration in the 'signal path' panel must be set correctly.



Select the frequency response correction file for this particular DBS2050A and click OK. The wave is then pre-compensated.

3. Once the wave is pre-compensated, it is displayed with the desired waveform on the Precompensation panel.

   If the amplitude is out of range (less than –127 DAC counts or greater than +127 DAC counts), the out of range indicator displays at the bottom of the panel. The data must be *clipped* or *scaled*.

   Select Clip or Scale on the panel, if the Amplitude is Out of Range, otherwise proceed to Step 4.

   If *clipped*, all data outside ±127 will become ±127.

   If *scaled*, the % of full scale indicator is calculated by:

   ((max-min)/254)*100.0

   The data is scaled by the function:

```
fsr=127;
if(abs(min_0)>max_0)max_0=abs(min_0);
scale = fsr/max_0;
data=data*scale;
```

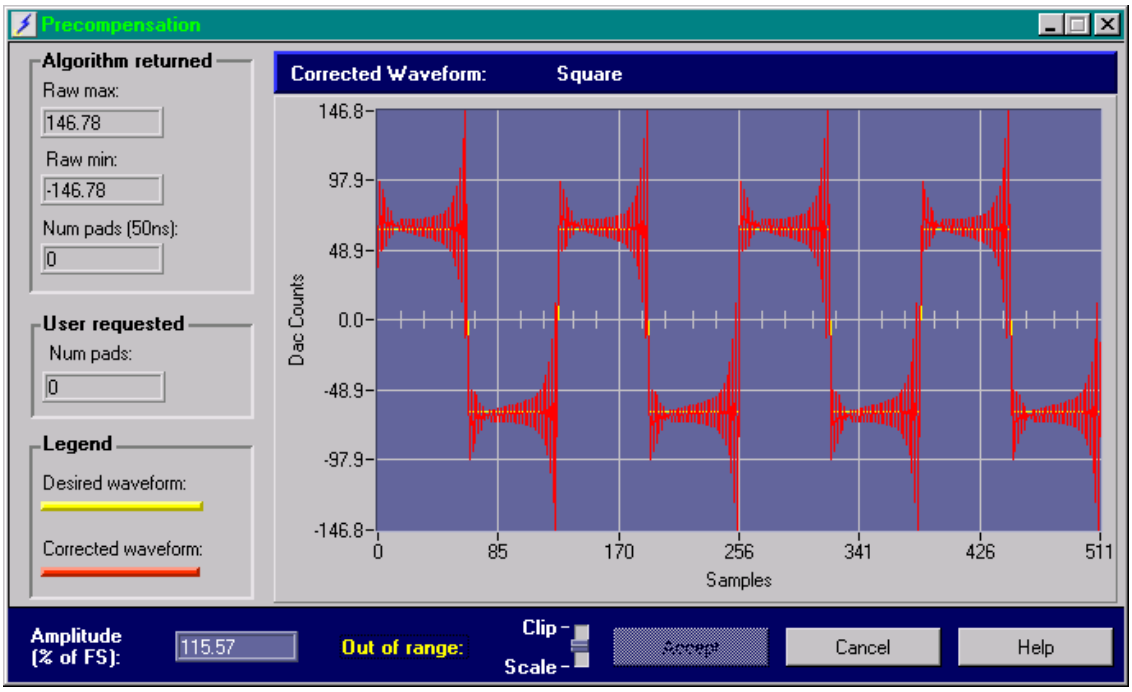


Figure 19: Precompensation panel

4. Click Accept to accept the pre-compensated wave and return to the Waveforms panel.

   Click Cancel to negate the pre-compensation and return to the Waveforms panel.

   Number of samples returned when in Pad Mode
   If pre-compensation occurred in Pad Mode, Precompensation panel will display how many pad samples were actually returned by the algorithm. If the algorithm returned more pad samples than requested, the SFP will only keep the number of samples requested. If the algorithm returns fewer pad samples than requested, the extra pad samples will remain 0. If the exact number of samples returned by the algorithm is desired, click Cancel to close the panel and repeat the pre-compensation process – this time entering the number of pad samples reported back the previous time.

Programming Notes: Calling the Pre-compensation .dll

Necessary Files

When programming, the following files are necessary:

- pre-complib.h – must be included
- pre-compLib.lib – must be used
- pre-compLib.dll – see below for function prototype description

Pre-compensation Function Prototype

```
Void _stdcall Pre-compensation
(float32 DesiredWaveform[],
int32 DesLen,
cmplx64 FreqRespCorrection[],
int32 FreqLen,
float32 InputFreqSpacing,
float32 SampleFreq,
uInt16 LoopOrPad,
uInt16 UnitType,
float32 CorrectedWaveform[],
int32 *CorrLen,
uInt32 *PadSamples,
float32 *MinOutput,
float32 *MaxOutput);
```

Parameter Descriptions

DesiredWaveform[ ]	Represents the Desired Waveform array.
DesLen	Represents the number of data points in the Desired Waveform.
FreqRespCorrection[ ]	Frequency response correction array (this can be obtained from the data file shipped on the floppy disk).
FreqLen	Represents the number of elements in the FreqRespCorrection array.
InputFreqSpacing	Frequency response correction frequency spacing - usually equaling 10e6.
SampleFreq	The Sample Clock frequency.
LoopOrPad	Loop Mode = 1, Pad Mode = 0
UnitType	DBS2050A = 0
CorrectedWaveform[ ]	Corrected waveform array returned by the pre-compensation algorithm.
***CorrLen**	Represents the length of the corrected waveform array.
***PadSamples**	Represents the number of pad samples returned (number added to beginning and end of waveform).
***MinOutput**	Represents the minimum value of data points in the corrected wave.
***MaxOutput**	Represents the maximum value of data points in the corrected wave.

Using the Pre-Compensation Library

Basic Rule to Follow when Pre-compensating a Waveform

When a waveform is pre-compensated, the resulting waveform is wider in time (50 ns on each end) than the original waveform, due to pad samples added to the beginning and end of the waveform. Depending upon sequence definition, additional manipulation of the pre-compensated waveform is necessary if looping is desired, or if multiple waveform segments are to be merged together to form one waveform sequence.

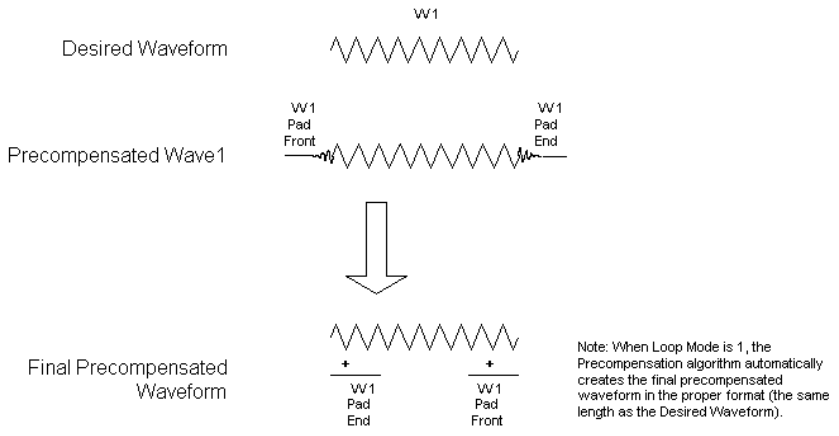
The additional manipulation involves simply adding the pad samples to the beginnings and/or ends of adjacent waveform segments, whenever there is an overlap, thereby bringing the waveforms back into proper time alignment. This is best understood graphically, as shown in the examples below.

Example: Single Segment Sequence

A single segment waveform can have two forms for the purpose of Pre-compensation.

Looping

This is where the single segment is repeated continuously as in a Free Run trigger mode. The LoopOrPad input should be set to 1 to define loop operation. In this case, the pre-compensation software takes care of any correction necessary and the Corrected waveform is equal to the Desired waveform length.

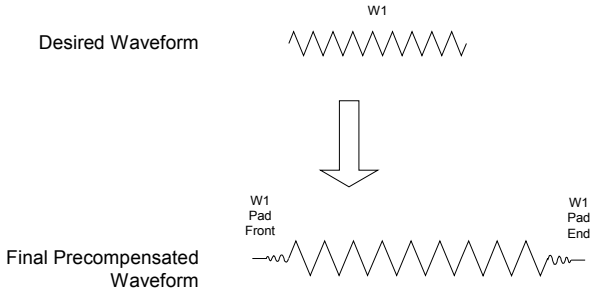


Single Segment Sequence: Looping Continuously

One Shot

This is where the single segment is run once after running or a trigger event. The LoopOrPad input should be set to zero to define pad operation. In this mode, the Corrected waveform is equal to the Desired

waveform length plus twice the Pad size defined by the PadSamples output. The number of Pad Samples can be reduced with loss of accuracy (see *Factors Affecting Accuracy*).



Single Segment Sequence: One Shot

(No Looping)

Multiple Segment Sequence

Pre-compensation on a multiple segment waveform can be performed in two ways:

Overall Pre-compensation

The waveform of the complete sequence with any looped segments flattened (i.e., N Loops converted to N segments) should be pre-compensated the same as for a Single Segment Sequence, as described above.

Multiple Segment Sequence

One Shot

Any individual segment is affected by the waveform in the adjacent segments. Thus to obtain the corrected waveform for any segment N we must pass segments N-1, N and N+1 separately through the Pre-compensation program with LoopOrPad= 0. The Pad samples at the start and end of segment N are removed (They can be used later for segments N-1 and N+1). The pad samples from the end of segment N-1 are added to the beginning samples of segment N in time order. (i.e. first pad sample adds to first segment value second pad sample to second segment value etc). The pad samples at the beginning of segment N+1 are added to the end of segment N in time order (i.e. Last pad sample adds to Last segment value second to last pad sample to second to last segment value etc). As a result of this operation, the segment N size is the same as the Desired segment N size.

Looping Continuously

The procedure is the same as One Shot except the pads are ‘wrapped,’ as shown at the bottom of the figure.

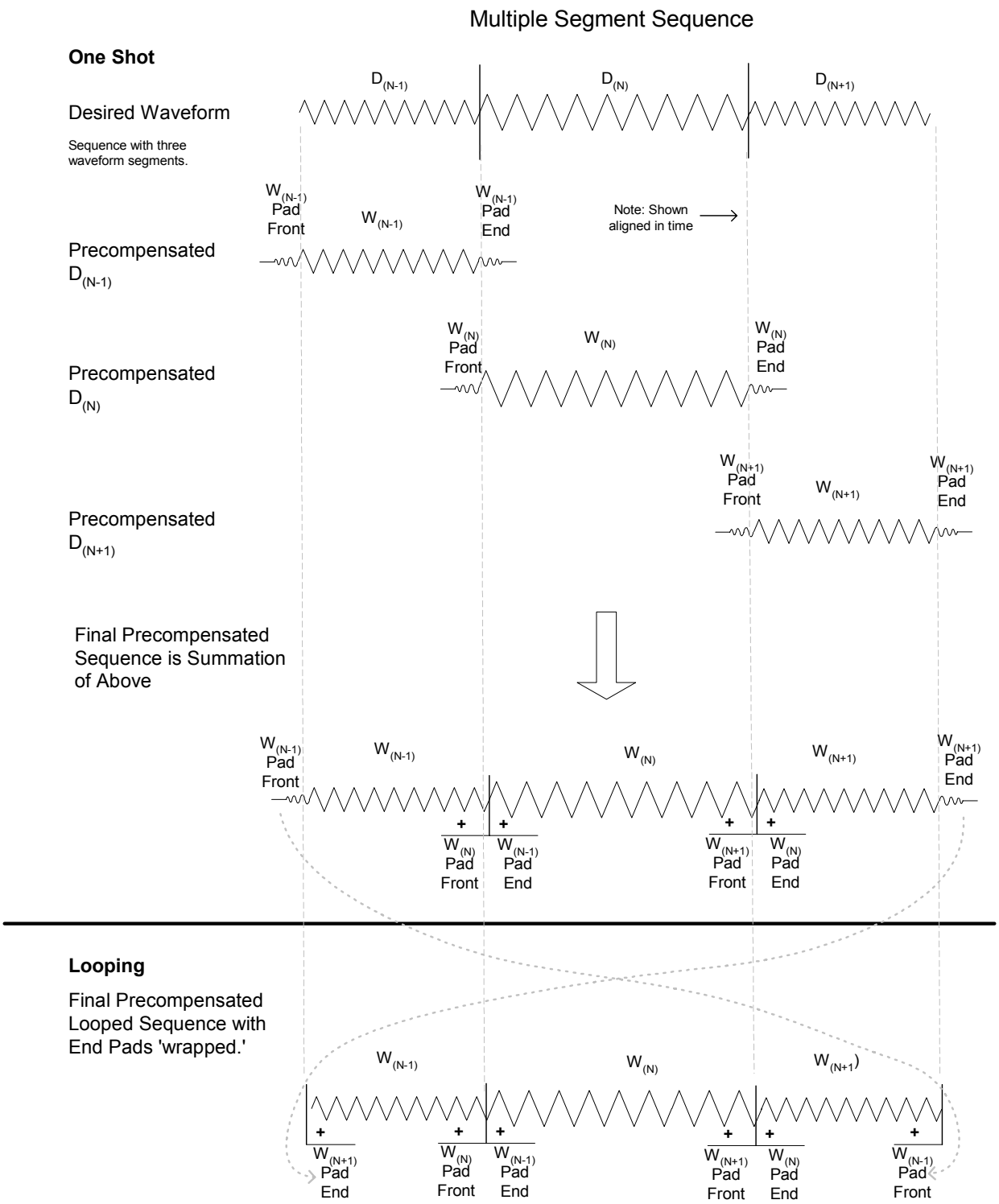
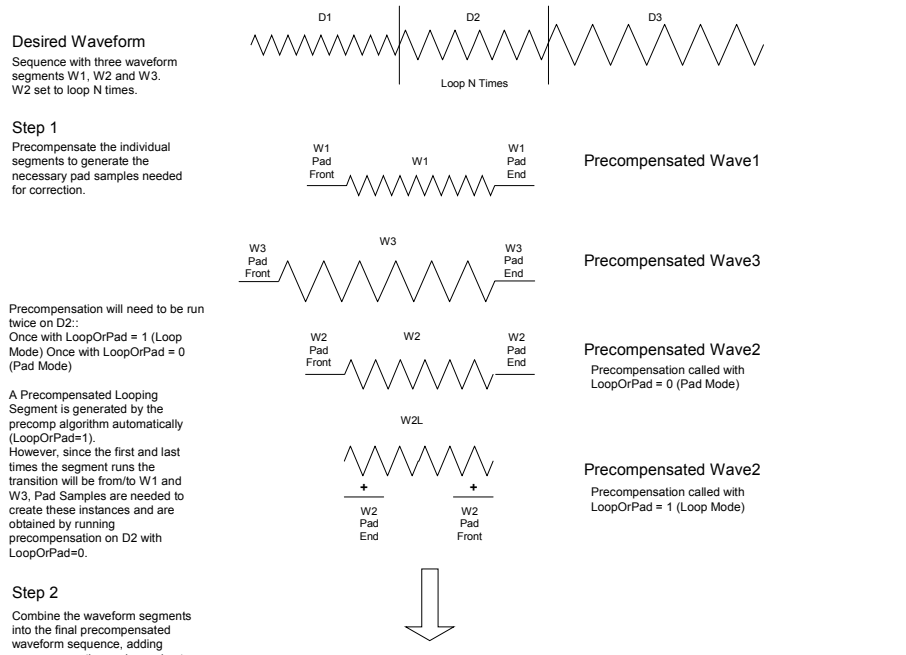


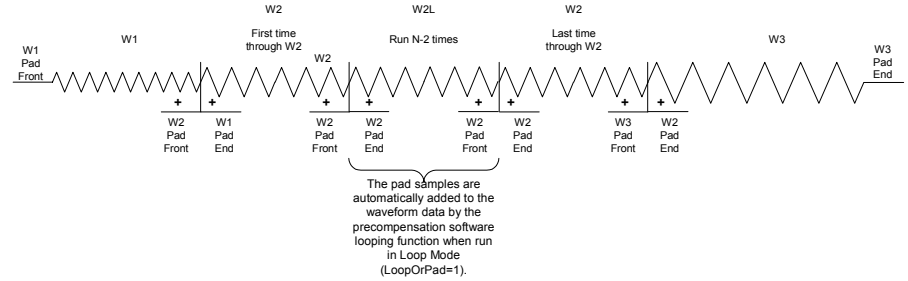
Figure 20: Multiple Segment Sequence

Multiple Segment Sequence – One Shot – with Looping of segments within the Sequence

Looping within a multiple segment structure is complicated by the fact that the beginning of the segment is affected by the previous segment when the loop segment is first entered and by the end of the looping segment on succeeding loops.



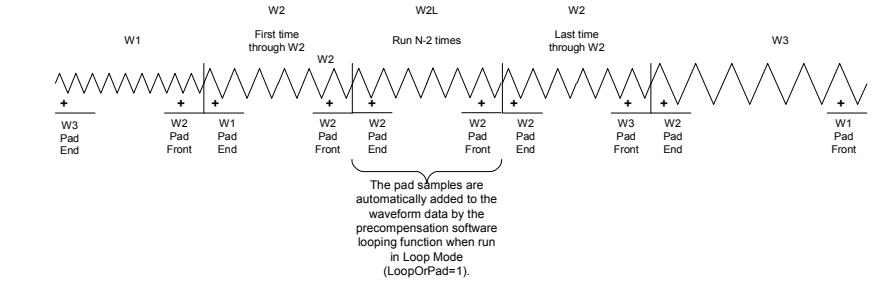
Final Precompensated Waveform



Multiple Segment Sequence: One Shot

With Looping of Individual Segment(s) Within

Final Precompensated Waveform



Multiple Segment Sequence: Looping Continuously

With Looping of Individual Segment(s) Within

Note that the only change between One Shot and Looping Continuously is to add the appropriate pad samples to the first and last segments.

Factors Affecting Accuracy

Reduction of the padding

The padding that is added to the beginning and end of the corrected waveform is the effect of the impulse response of the Frequency Response Correction data. The extent of the padding is 1/(2* Input Freq Spacing). This is 50ns for factory supplied Frequency Response Correction data (Input Freq Spacing = 10MHz). The algorithm assumes that the Desired Waveform is zero over this time.

This padding can be truncated but will cause an error in the Output Waveform after the point of truncation. The permissible amount of truncation is obtained by experiment.

Clipping

As the Frequency Response Correction is always greater than one and increasing with frequency, the Corrected Waveform will be larger in amplitude than the Desired waveform.

Thus if a Desired waveform fills 100% of the range of the DAC the Corrected Waveform will be larger than 100% and if applied to the DAC will display clipping of the waveform when the value goes outside the DAC range.

To avoid clipping, the Desired Waveform must be lowered in amplitude or offset (if clipping is asymmetrical) by the appropriate amount.

The choice whether to allow clipping or not is left up to the user. Clipping will cause an error in the instrument output just after the clipped waveform. This might be acceptable in many instances.

To minimize the amount of clipping, any unnecessary high frequency content in the Desired Waveform should be minimized. Examples of waveforms generating high frequency content are single sample transitions of large amplitude and high amplitude high frequency oscillations.

Signal Bandwidth

The effective bandwidth of the Corrected output is 0.5*Fs for the DBS2050A. Fs= Sample Frequency. The shape of the response is 'Brick wall' (flat to cut-off and then zero above) in the frequency domain and having a SINC impulse response.

For accurate reproduction of the Desired Waveform its frequency spectrum should lie below the 0.5*Fs. Any components above the cutoff will be eliminated or aliased.

Aliasing

As in any sampled system, alias components exist. These are represented in the frequency domain by a folding of the signal spectrum about the sampling frequency. Any Desired Waveform that has frequency components above Nyquist (0.5*Fs) will see these components overlapped on the spectrum below Nyquist, thus creating errors in the output waveform.

The DBS2050A is supplied with a reconstruction filter that cuts off just below Nyquist for Fs= 2.4GHz). This filter is designed to filter out the alias components above Nyquist.

Sampling Frequency Limitations

For optimum performance of the DBS2050A, the sample frequency (Fs) should be 2.4GHz. This matches the supplied filter. If Fs is lowered then depending on the frequency content of the signal, more alias energy will appear in the desired spectrum.

Any waveform with significant signal frequency content less than 1.2GHz for DBS2050A can be reproduced very accurately using Pre-compensation with a sample frequency of 2.4GHz. The only reason to reduce the sample frequency would be if the waveform length exceeds the available waveform memory. Under this circumstance, a lower sampling frequency must be used with subsequent waveform error.

Stimulus/Measurement Instrumentation
A Family of Test Instruments

ANALOGIC

Document #82-5126
Revision 03

Proprietary Statement

The information contained in this publication is derived in part from proprietary and patent data of the Analogic Corporation. This information has been prepared for the express purpose of assisting operating and maintenance personnel in the efficient use of the instrument described herein. Publication of this information does not convey any rights to reproduce it or to use it for any purpose other than in connection with the installation, operation, and maintenance of the equipment described herein.

Printed in U.S.A.

Contents

PROPRIETARY STATEMENT II

1 INTRODUCTION 1-1

GENERAL 1-1
THE STIMULUS/MEASUREMENT SYSTEM 1-1
Platform Test Sets 1-1
VXI-Compatible Bench Test Sets 1-2
CONFIGURING A STIMULUS/TEST SYSTEM 1-2
ABBREVIATIONS AND SYMBOLS 1-5
BOOK PLAN 1-6

2 PLATFORMS 2-1

GENERAL 2-1

3 CARRIERS 3-1

GENERAL 3-1
CARRIER PERFORMANCE 3-1
Interfacing 3-1
Clocking and Triggering 3-1
CARRIER CONFIGURATIONS 3-2

4 INSTRUMENT MODULES 4-1

GENERAL 4-1

5 SOFTWARE 5-1

GENERAL 5-1
WAVESMITH© 5-1
WAVESMITH PLUS© 5-1
PLATINUM 5-1
PRE-COMPENSATION 5-2
SOFT FRONT PANEL 5-2

6 CONFIGURATIONS FOR TYPICAL APPLICATIONS 6-1

GENERAL 6-1
FAMILY TREE 6-2

Illustrations

Figure 1A. Top-Level VXI Bench and Platform Family Trees 1-3
Figure 1B. Carrier and Instrument Modules, Continuation of Family Trees 1-4
Figure 2. Platforms DP7020-2 and DP7040-2 2-1
Figure 3. Carrier Assembly for Two Instrument Modules 3-2
Figure 4. Carrier with Modules DBS901 and DBS902 3-2
Figure 5. Family Tree for Test System Application 6-2

1
Introduction

General

Analogic offers a complete product line of platforms' carriers, instruments, modules, circuit boards, and software. These are used to configure high-speed, high-precision stimulus/measurement test systems for a broad range of applications in the development laboratory and on the factory floor.

This Addendum includes a brief description of the product line, including software packages[1], It also includes examples of test systems that may be configured for typical Stimulus/Measurement applications. Details of Installation, Operation, and Maintenance of each product are contained in separate instruction manuals for those products; such as DBS2050A*, 2.4-GS/s Dual-Channel Arbitrary Waveform Generator*, *User's Manual*

NOTE: Information included in this addendum describes options and accessories that may not be installed in your equipment. Consult your configuration listing for those that apply to your unit.

The Stimulus/Measurement System

As shown in Figure 1A, there are two main streams of Stimulus / Measurement Test Instruments. Each stream contains a combination of hardware and software units to generate electronic stimulus signals and to measure output electronic responses. They provide the user with a family of compatible hardware and software units from which to configure a cost-effective, application-specific Stimulus /Measurement Test System.

Platform Test Sets

Whatever the final configuration, each platform test system hardware assembly contains one of two platforms, as shown in Figure 1A, one of two carriers, and a selection of instrument modules, as shown in Figure 1B. The DBS2050A and DBS2055 incorporate the instrument module functions within their design.

[1] Consult the factory for product delivery schedule and pricing.

Two software packages (Wavesmith and Drivers) are provided as standard units for each platform test system. Three optional software packages, Wavesmith Plus, NRB, and Pre-Compensation are available.

VXI-Compatible Bench Test Sets

As shown in Figure 1A, the VXI-Compatible Bench Test Set hardware is essentially a subset of that of the Platform Systems; namely, the Carrier and Instrument Module units. VXI-compatible Test System hardware assembly will contain one of two carriers and a selection of instrument modules, as shown in Figure 1B.

Plug-and-Play Drivers and Soft Front Panel (SFP) software are provided for each VXI-Compatible Test System assembly.

Configuring a Stimulus/Test System

In a typical configuring exercise, the user starts with the selection of the module (or carrier with module of DBS20xx) and then selects the appropriate carrier according to the Module Selection. Additional configuring information is included in Chapter 6 *Applications*.

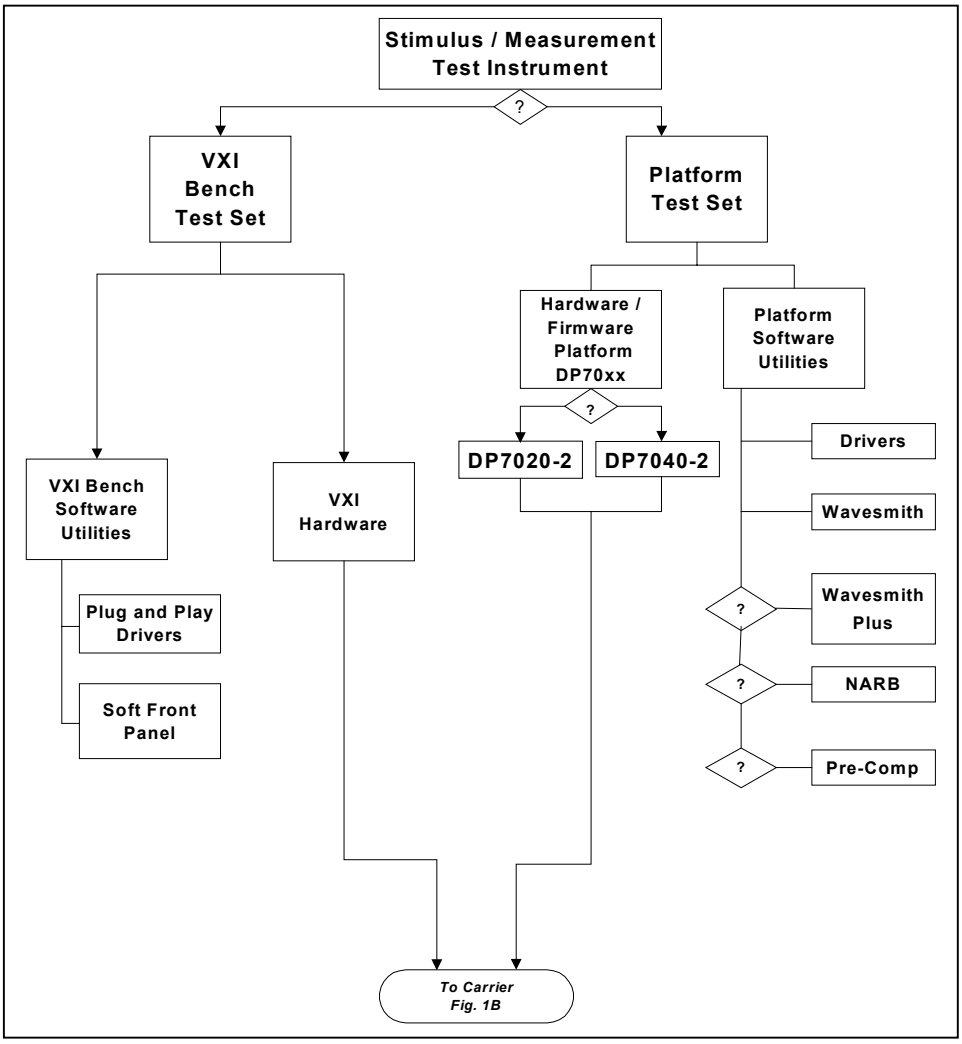


Figure 1A. Top-Level VXI Bench and Platform Family Trees

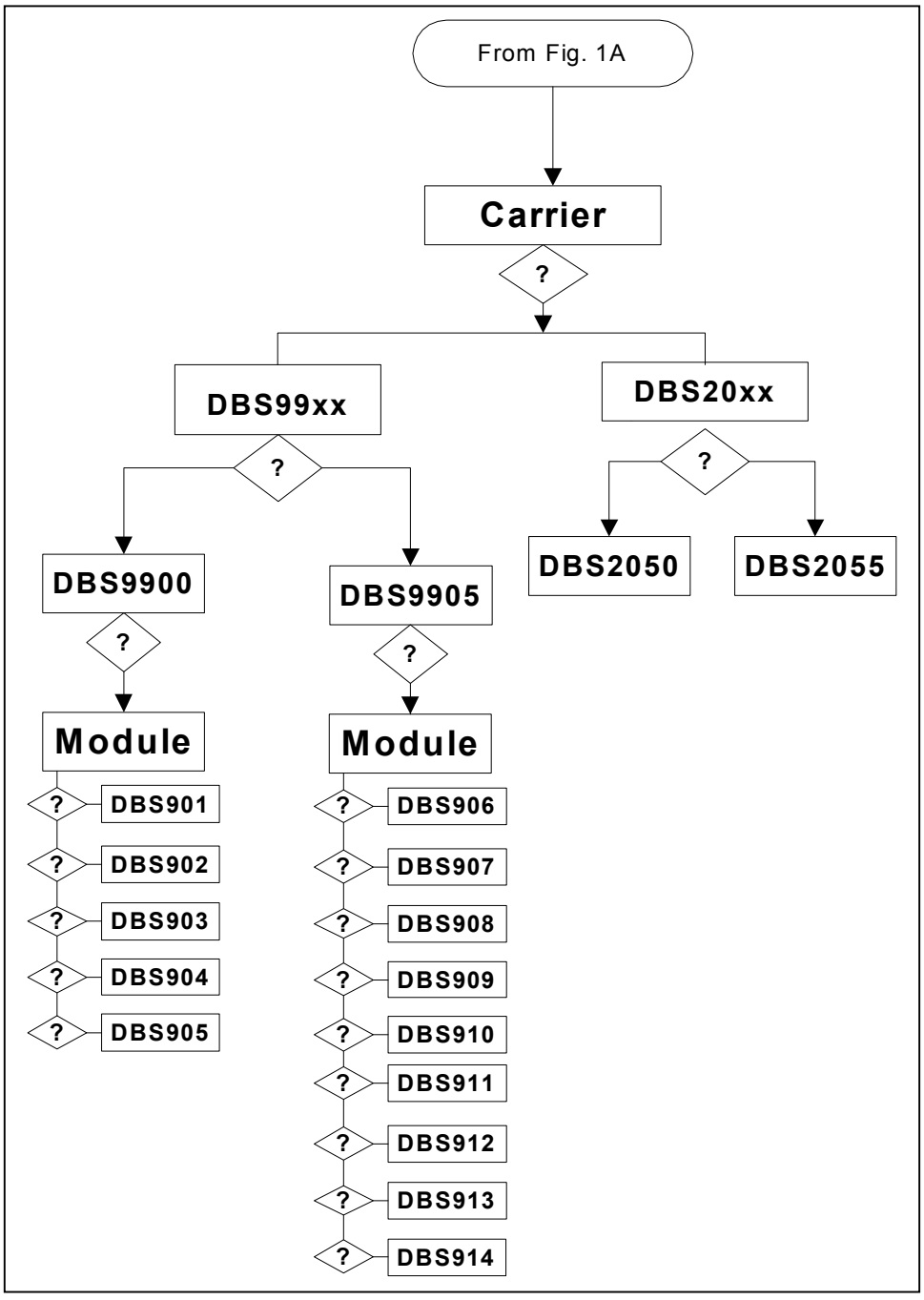


Figure 1B. Carrier and Instrument Modules, Continuation of Family Trees

Abbreviations and Symbols

Various abbreviations and symbols are used throughout this Addendum and in the separate instruction manuals. They are explained when first used. However, a consolidated listing of these terms is included in each document.

Abbreviation	Meaning
A/D	Analog-to-Digital Converter' (See, also, ADC)
ADC	Analog-to-Digital Converter
ARB	Arbitrary waveform. See also AWG
AWG	Arbitrary Waveform Generator
CPCI	Compact PCI
D/A	Digital-to-Analog Converter (See, also, DAC)
DAC	Digital-t0-Analog Converter
DIG	Digitizer
G S/s	Giga-samples per second
GHz	Giga Hertz (10^9 cycles per second) unit of frequency
GPIB	General Purpose Instrument Bus
K S/s	Kilo-samples per second
KHz	Kilo Hertz (1000 cycles per second); unit of frequency
M S/s	Mega-samples per second
MHz	Mega Hertz (1,000,000 cycles per second); unit of \frequency
NARB	Non-Arbitrary Waveform
PCB	Printed Circuit Board
RS232	IEEE standard for serial I/O
SFP	Soft Front Panel
1394	IEEE standard for Fire Wire interface

Book Plan

The remaining chapters in this Addendum are:

Chapter 2, *Platforms,* containing descriptions of features and of mechanical/physical and performance specifications common to the available platforms. It also includes the features unique to each platform in the family.

Chapter 3, *Carriers,* containing descriptions of features and of mechanical/physical and performance specifications common to the available carriers. It also includes the features unique to each carrier in the family.

Chapters 4, *Instrument Modules*, containing descriptions of the instruments that are currently available for installation in a designated carrier. It includes mechanical/physical and performance specifications common to the instruments, as well as the distinctive features of each.

Chapter 5, *Software*, containing descriptions of the available software packages for generating test waveforms and for measuring responses It contains a brief description of the standard and optional software available for bech test systems and for VXI-compatible test systems.

Chapter 6, *Applications,* containing configurations that may be assembled for typical waveform generating and waveform digitizing systems. It includes illustration of the family tree (as previously illustrated in Figure 1) for a typical application-specific system.

2
Platforms

General

There are two platform models: DP7020-2 and DP7040-2. Each platform incorporates an embedded controller, power supply, On-Off power switch, fan, and front-panel status display LEDs. See Figure 1.

Analogic Model #	Controller/Host Interface	Instrument Slots	Power Rating
DP7020-2	Embedded[2]	2	300 Watts
DP7040-2	Embedded[2]	4	600 Watts

Figure 2. Platforms DP7020-2 and DP7040-2

[2] 1394 Fire Wire, GPIB, and 100 BT Ethernet are included.

3
Carriers

General

There are two Carriers currently available in the Analogic product line: Models DBS9900 and DBS9905. The DBS9900 is intended for use in wave generating and digitizing systems requiring applications up to 500 MS/s. The DBS9905 is intended for use in systems requiring applications above 500 MS/s.

Both Analogic Carriers are VXI C-size compliant for multiple Instrument Modules. They accept one or two modules from the DBS 9xx family of high-speed, high precision waveform generators and digitizers. Each module contains the circuitry necessary to operate independently for a complete channel of analog input or output. This design allows a single VXI chassis slot to provide multiple functions.

Carrier Performance

Major performance features are described in this paragraph. Detailed specifications are contained in the Carrier Instruction Manual.

Interfacing

The entire VXI interface is implemented in a single device (Platinum) providing increased reliability and lower cost. The Carrier contains the basic register set for a register-based VXI device and also the CLASS Dependent Register set (used for functions common to all modules resident on the DBS9900).

Clocking and Triggering

Each Carrier provides instrumentation for both internal and external triggering. Versatile clocking and triggering is provided via internal circuitry. Front panel connectors accept VXI TTLTRG or ECLTRG lines. The internal clock may be used to drive all plug-in function modules simultaneously so that clock skew and delays are minimized. This allows coherent operation of multiple plug-in modules. Programmable trigger and clock thresholds are provided for each module.

Carrier Configurations

Figure 3 illustrates the Assembly of a typical Carrier. Front-panel labels identifying the installed modules are affixed on the Carrier extractors.

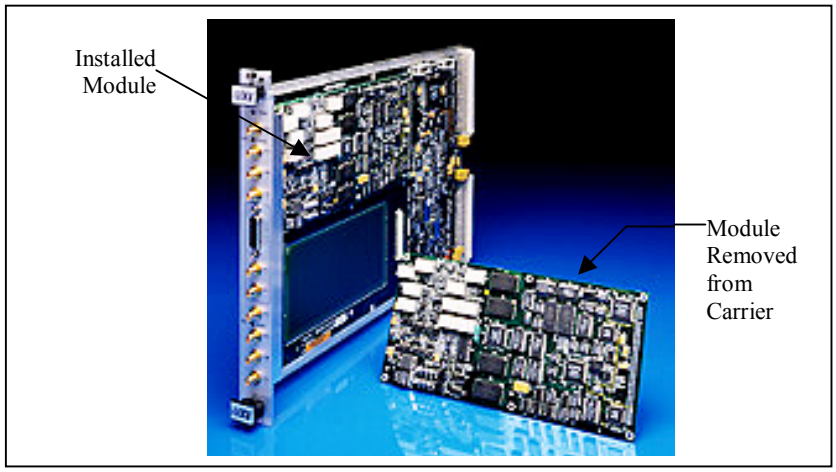
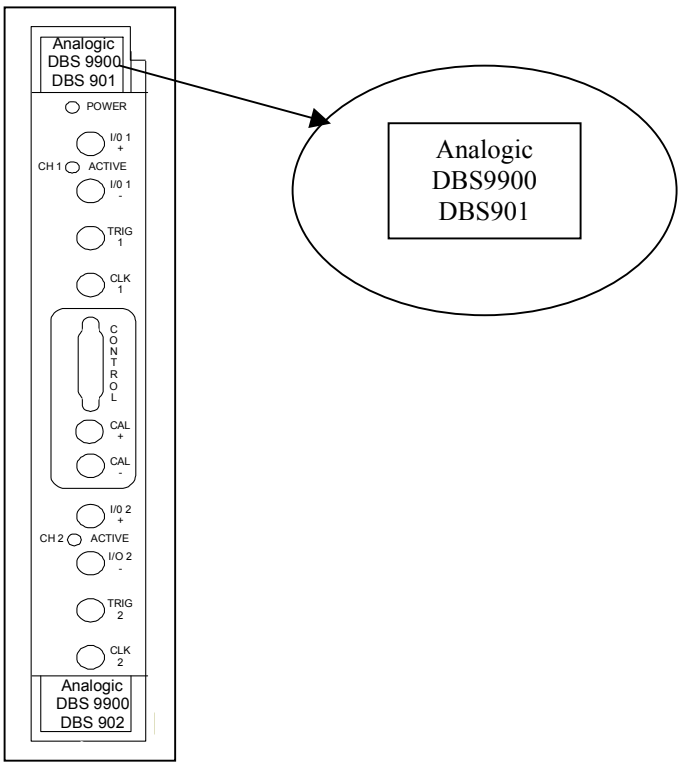


Figure 3. Carrier Assembly for Two Instrument Modules

Figure 4 illustrates a Carrier front panel with labels indicating installed



modules DBS901 (Top -- Module 1) and DBS902 (Lower -- Module 2).

Figure 4. Carrier with Modules DBS901 and DBS902

4
Instrument Modules

General

The table that follows indicates the significant performance specifications of instrument modules that may be installed in each of the two available carriers. This chapter provides information of the major performance features of each group.

Analogic Carrier Model #	Instrument Module Name	Type	Speed	Bandwidth	Resolution
DBS9900	DBS901	AWG	100 MS/s	50 MHz	14-Bit
	DBS902	DIG	80 MS/s	40 MHz	14-Bit
	DBS903	AWG	4 MS/s	1 MHz	22-Bit
	DBS904	DIG	4 MS/s	1 MHz	22-Bit
	DBS905	AWG	500 MS/s	250 MHz	12-Bit
DBS9905	DBS906	DIG	500 MS/s	125 MHz	12-Bit
	DBS907	DIG	1 GS/s	250 MHz	8-Bit
	DBS908	DIG	2 GS/s	500 MHz	8-Bit
	DBS909	DIG	4 GS/s	1 GHz	8-Bit
	DBS910	AWG	4 GS/s	2 GHz	8-Bit
	DBS911	AWG	8 GS/s	4 GHz	10-Bit
	DBS912	DIG	8 GS/s	2 GHz	10-Bit
	DBS913	DIG	10 MS/s	2.5 GHz	14-Bit
	DBS914	DIG	200 KS/s	5 GHz	16-Bit
DBS2050A		AWG	2.4 GS/s	950 MHz* 1.2 GHz**	8-Bit
DBS2055		AWG	4.8 GS/s	950 MHz* 2 GHz**	8-Bit

The DBS2050A and DBS2055 are assembled within their own carriers.

* Without Waveform Precompensation Installed
** With Waveform Precompensation Installed

Features:

- A programmable threshold External Clock can be supplied to each module through front panel connectors (labeled CLK1 and CLK2) to control the sample rate.
- An internal PLL based sample clock source can be supplied to each module.
- An external 10MHz reference clock can be supplied to the internal clock.
- A programmable threshold External Trigger is also supplied for each module through front panel connectors (labeled TRIG1 and TRIG2) to initiate activity of that module.
- Any one of eight VXI TTLTRIG lines can be selected as an external back-plane trigger. These can be used with the Internal or External Clock. One of two ECL back-plane trigger lines can also be selected.

5
Software

General

This section contains descriptions of the major software packages, available for use with the Analogic product family of Waveform Generators. They are:

- Wavesmith and Wavesmith Plus,
- Platinum
- Pre-Compensation
- Soft Front Panel

Wavesmith©

The basic operations of Wavesmith are:

1. Create a "blank" wave
2. Use various Wavesmith tools to customize the wave for a particular design or test function in a format which is compatible with a particular arbitrary waveform generator,
3. Save it to a file and/or transfer it to the waveform generator, and
4. Perform basic analysis on streams of data.

Wavesmith Plus©

Wavesmith Plus is project oriented. Using object icons of Wavesmith Plus, the programmer can set up a system of processing incoming data and store that system as one of the computer program modules.

Platinum

Platinum is a firmware device containing interfacing protocols that convert incoming electrical signals to and from VXI-compatible format. The I/O formats that are converted in Platinum are:

- IEEE1394 Fire Wire
- RS232

- Ethernet 10T/100T

Pre-Compensation

The Pre-Compensation software package operates on the frequency components of a generated waveform to provide an essentially flat frequency distribution out to a selected 3-dB point. This enables the test stimulus to stress the UUT (Unit Under Test) with equal levels throughout the frequency range.

Soft Front Panel

The Soft Front Panel provides a GUI capability with which to develop and program the generated waveform.

6
Configurations for Typical Applications

General

As described briefly in Chapter 1 *Introduction*, configuring an application-specific test system begins with the selection of processing modules with the parameter values required for that operation. The chart that follows indicates selections that may be made to configure a Test System for each of five applications.

Application Description	Platform	Carrier	Instrument Module(s)	Optional Software
1. Stress Test of Disk Drive Read and Write Channels	DP7040-2	DBS2055		NARB
2. Test of Synchronous Generation of I and Q phases of Quadrature Generation	DP7040-2	DBS2050A (2)		Wavesmith Wavesmith Plus
3. Production test of Disk Drive Read/Write Channel Printed Circuit Board (PCB)	DP7020-2	DBS2050A		NARB Wavesmith
4. General Purpose Testing and Measuring at Frequencies up to 80MHz	DP7020-2	DBS9900	DBS901 DBS902	Wavesmith Wavesmith Plus
5. General Purpose Testing and Measuring at Frequencies above 500 MHz	DP7020-2	DBS9905	DBS907 DBS908	Wavesmith Wavesmith Plus
6. Quadrature Testing	DP7020-2	DBS9900	DBS901 DBS901	Wavesmith Wavesmith Plus
7. R&D Development of Test for DSL	DP7020-2	DBS9900	DBS901 DBS902	Wavesmith Wavesmith Plus
8. Testing of Pacemakers and Defibrillators	DP7020-2	DBS9900	DBS902	Wavesmith Wavesmith Plus

As shown in the table, the same Test/Measurement System may be used in more than one application.

Family Tree

The configuration Family Tree for Application #4 is shown in Figure 5.

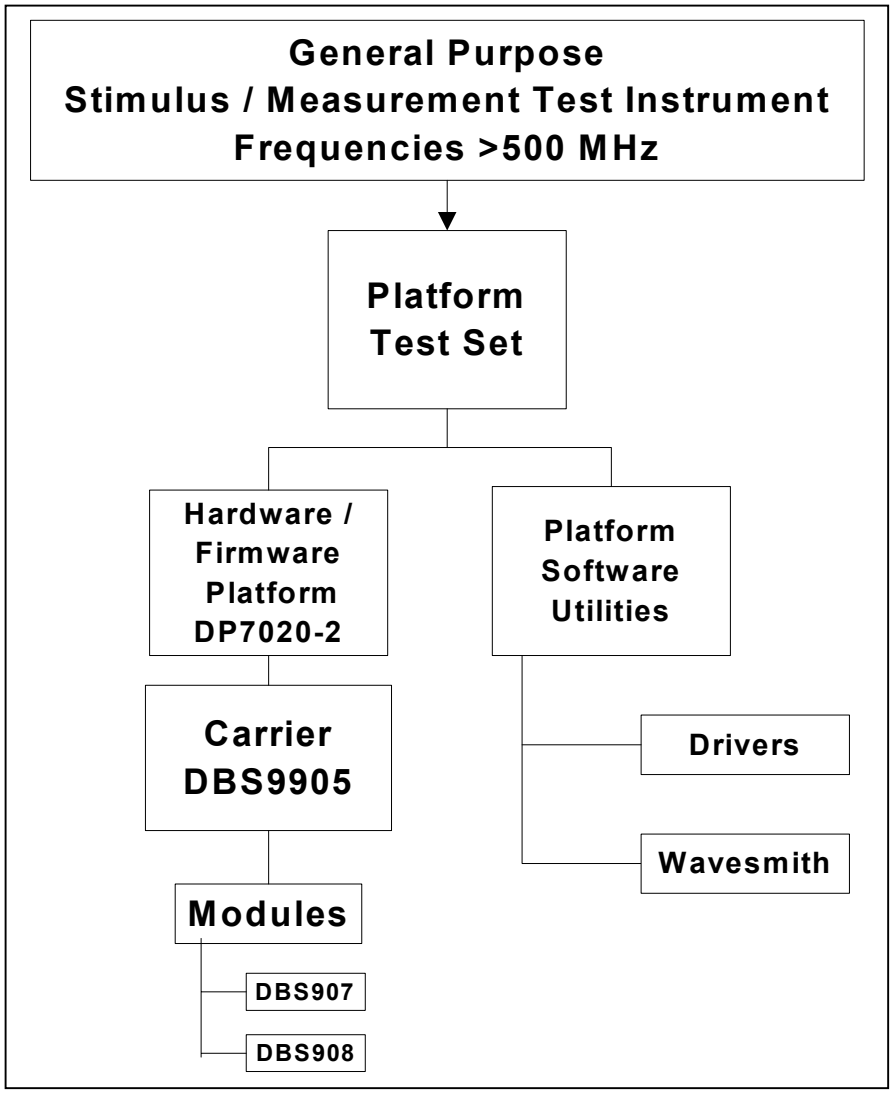


Figure 5. Family Tree for Test System Application

Analogic Corporation
Test & Measurement Division
8 Centennial Drive
Peabody, MA 01960-7987, USA
Tel: (978) 977-3000
Fax: (978) 977-6814
email: t&m_info@analogic.com
www.analogic.com

P/N 82-5126 Rev. 03

ANALOGIC®
The World Resource for Precision Signal Technology